MyBatis 中文文档 3.4

wizardforcel

Published with GitBook



目錄

- 介紹 0
- 参考文档 1
  - 简介 1.1
  - 入门 1.2
  - XML 映射配置文件 1.3
  - Mapper XML 文件 1.4
  - 动态 SQL 1.5
  - Java API 1.6
  - SQL语句构建器类 1.7
  - Logging 1.8
- 项目文档 2
  - 项目总体信息 2.1
    - 访问 2.1.1
    - 提醒方法 2.1.2
    - 项目依赖 2.1.3
    - Dependency Information 2.1.4
    - Overview 2.1.5
    - 问题跟踪 2.1.6
    - 项目授权 2.1.7
    - 项目邮件列表 2.1.8
    - Project Plugin Management 2.1.9
    - Project Build Plugins 2.1.10
    - Project Report Plugins 2.1.11
    - 团队 2.1.12
    - Web访问 2.1.13
    - 匿名访问 2.1.14
    - 开发者访问 2.1.15
    - 通过防火墙访问 2.1.16
    - 项目概要 2.1.17
  - 生成报表 2.2

MyBatis 中文文档 3.4

来源：MyBatis中文文档

参考文档

简介

什么是 MyBatis ？

MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以对配置和原生Map使用简单的 XML 或注解，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。

帮助改进文档...

不管你以何种方式发现了文档的不足，或是丢失对某一特性的描述，那么你能做的最好的事情莫过于去研究它并把文档写出来。

该文档 xdoc 格式的源码文件可通过项目的 Git 代码库来获取。Fork 该源码库，做出更新，然后提交一个 pull request 吧。

你将成为本文档的最佳作者，MyBatis 的用户定会过来查阅的。

当前的国际化版本

MyBatis 的其他语言版本：

- English
- Espa?ol
- 日本語
- ???
- 简体中文

你想使用本地语言来了解MyBatis吗？那就将它翻译成你的母语并提供给我们吧！

入门

安装

要使用 MyBatis， 只需将 `mybatis-x.x.x.jar` 文件置于 classpath 中即可。

如果使用 Maven 来构建项目，则需将下面的 dependency 代码置于 pom.xml 文件中：

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>x.x.x</version>
</dependency>
```

从 XML 中构建 SqlSessionFactory

每个基于 MyBatis 的应用都是以一个 SqlSessionFactory 的实例为中心的。SqlSessionFactory 的实例可以通过 SqlSessionFactoryBuilder 获得。而 SqlSessionFactoryBuilder 则可以从 XML 配置文件或一个预先定制的 Configuration 的实例构建出 SqlSessionFactory 的实例。

从 XML 文件中构建 SqlSessionFactory 的实例非常简单，建议使用类路径下的资源文件进行配置。但是也可以使用任意的输入流(InputStream)实例，包括字符串形式的文件路径或者 file:// 的 URL 形式的文件路径来配置。MyBatis 包含一个名叫 Resources 的工具类，它包含一些实用方法，可使从 classpath 或其他位置加载资源文件更加容易。

```
String resource = "org/mybatis/example/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStrea
```

XML 配置文件（configuration XML）中包含了对 MyBatis 系统的核心设置，包含获取数据库连接实例的数据源（DataSource）和决定事务范围和控制方式的事务管理器（TransactionManager）。XML 配置文件的详细内容后面再探讨，这里先给出一个简单的示例：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

当然，还有很多可以在XML 文件中进行配置，上面的示例指出的则是最关键的部分。要注意 XML 头部的声明，用来验证 XML 文档正确性。environment 元素体中包含了事务管理和连接池的配置。mappers 元素则是包含一组 mapper 映射器（这些 mapper 的 XML 文件包含了 SQL 代码和映射定义信息）。

不使用 XML 构建 SqlSessionFactory

如果你更愿意直接从 Java 程序而不是 XML 文件中创建 configuration，或者创建你自己的 configuration 构建器，MyBatis 也提供了完整的配置类，提供所有和 XML 文件相同功能的配置项。

```
DataSource dataSource = BlogDataSourceFactory.getBlogDataSource();
TransactionFactory transactionFactory = new JdbcTransactionFactory(
Environment environment = new Environment("development", transactic
Configuration configuration = new Configuration(environment);
configuration.addMapper(BlogMapper.class);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder(
```

注意该例中，configuration 添加了一个映射器类（mapper class）。映射器类是 Java 类，它们包含 SQL 映射语句的注解从而避免了 XML 文件的依赖。不过，由于 Java 注解的一些限制加之某些 MyBatis 映射的复杂性，XML 映射对于大多数高级映射（比如：嵌套 Join 映射）来说仍然是必须的。有鉴于此，如果存在一个对等的 XML 配置文件的话，MyBatis 会自动查找并加载它（这种情况下，BlogMapper.xml 将会基于类路径和 BlogMapper.class 的类名被加载进来）。具体细节稍后讨论。

从 SqlSessionFactory 中获取 SqlSession

既然有了 SqlSessionFactory，顾名思义，我们就可以从中获得 SqlSession 的实例了。SqlSession 完全包含了面向数据库执行 SQL 命令所需的所有方法。你可以通过 SqlSession 实例来直接执行已映射的 SQL 语句。例如：

```
SqlSession session = sqlSessionFactory.openSession();
try {
  Blog blog = (Blog) session.selectOne("org.mybatis.example.BlogMap
} finally {
  session.close();
}
```

诚然这种方式能够正常工作，并且对于使用旧版本 MyBatis 的用户来说也比较熟悉，不过现在有了一种更直白的方式。使用对于给定语句能够合理描述参数和返回值的接口（比如说BlogMapper.class），你现在不但可以执行更清晰和类型安全的代码，而且还不用担心易错的字符串字面值以及强制类型转换。

例如：

```
SqlSession session = sqlSessionFactory.openSession();
try {
  BlogMapper mapper = session.getMapper(BlogMapper.class);
  Blog blog = mapper.selectBlog(101);
} finally {
  session.close();
}
```

现在我们来探究一下这里到底是怎么执行的。

探究已映射的 SQL 语句

现在，或许你很想知道 SqlSession 和 Mapper 到底执行了什么操作，而 SQL 语句映射是个相当大的话题，可能会占去文档的大部分篇幅。不过为了让你能够了解个大概，这里会给出几个例子。

在上面提到的两个例子中，一个语句应该是通过 XML 定义，而另外一个则是通过注解定义。先看 XML 定义这个，事实上 MyBatis 提供的全部特性可以利用基于 XML 的映射语言来实现，这使得 MyBatis 在过去的数年间得以流行。如果你以前用过 MyBatis，这个概念应该会比较熟悉。不过 XML 映射文件已经有了很多的改进，随着文档的进行会愈发清晰。这里给出一个基于 XML 映射语句的示例，它应该可以满足上述示例中 SqlSession 的调用。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = #{id}
  </select>
</mapper>
```

对于这个简单的例子来说似乎有点小题大做了，但实际上它是非常轻量级的。在一个 XML 映射文件中，你想定义多少个映射语句都是可以的，这样下来，XML 头部和文档类型声明占去的部分就显得微不足道了。文件的剩余部分具有很好的自解释性。在命名空间“com.mybatis.example.BlogMapper”中定义了一个名为“selectBlog”的映射语句，这样它就允许你使用指定的完全限定名“org.mybatis.example.BlogMapper.selectBlog”来调用映射语句，就像上面的例子中做的那样：

```
Blog blog = (Blog) session.selectOne("org.mybatis.example.BlogMappe
```

你可能注意到这和使用完全限定名调用 Java 对象的方法是相似的，之所以这样做是有原因的。这个命名可以直接映射到在命名空间中同名的 Mapper 类，并在已映射的 select 语句中的名字、参数和返回类型匹配成方法。这样你就可以向上面那样很容易地调用这个对应 Mapper 接口的方法。不过让我们再看一遍下面的例子：

```
BlogMapper mapper = session.getMapper(BlogMapper.class);
Blog blog = mapper.selectBlog(101);
```

第二种方法有很多优势，首先它不是基于字符串常量的，就会更安全；其次，如果你的 IDE 有代码补全功能，那么你可以在有了已映射 SQL 语句的基础之上利用这个功能。

提示命名空间的一点注释

命名空间（**Namespaces**）在之前版本的 MyBatis 中是可选的，容易引起混淆因此是没有益处的。现在的命名空间则是必须的，目的是希望能比只是简单的使用更长的完全限定名来区分语句更进一步。

命名空间使得你所见到的接口绑定成为可能，尽管你觉得这些东西未必用得上，你还是应该遵循这里的规定以防哪天你改变了主意。出于长远考虑，使用命名空间，并将它置于合适的 Java 包命名空间之下，你将拥有一份更加整洁的代码并提高了 MyBatis 的可用性。

命名解析：为了减少输入量，MyBatis 对所有的命名配置元素（包括语句，结果映射，缓存等）使用了如下的命名解析规则。

- 完全限定名（比如“com.mypackage.MyMapper.selectAllThings”）将被直接查找并且找到即用。
- 短名称（比如“selectAllThings”）如果全局唯一也可以作为一个单独的引用。如果不唯一，有两个或两个以上的相同名称（比如“com.foo.selectAllThings ”和“com.bar.selectAllThings”），那么使用时就会收到错误报告说短名称是不唯一的，这种情况下就必须使用完全限定名。

对于像 BlogMapper 这样的映射器类（Mapper class）来说，还有另一招来处理。它们的映射的语句可以不需要用 XML 来做，取而代之的是可以使用 Java 注解。比如，上面的 XML 示例可被替换如下：

```
package org.mybatis.example;
public interface BlogMapper {
  @Select("SELECT * FROM blog WHERE id = #{id}")
  Blog selectBlog(int id);
}
```

对于简单语句来说，注解使代码显得更加简洁，然而 Java 注解对于稍微复杂的语句就会力不从心并且会显得更加混乱。因此，如果你需要做很复杂的事情，那么最好使用 XML 来映射语句。

选择何种方式以及映射语句的定义的一致性对你来说有多重要这些完全取决于你和你的团队。换句话说，永远不要拘泥于一种方式，你可以很轻松的在基于注解和 XML 的语句映射方式间自由移植和切换。

范围（Scope）和生命周期

理解我们目前已经讨论过的不同范围和生命周期类是至关重要的，因为错误的使用会导致非常严重的并发问题。

提示 对象生命周期和依赖注入框架

依赖注入框架可以创建线程安全的、基于事务的 SqlSession 和映射器（mapper）并将它们直接注入到你的 bean 中，因此可以直接忽略它们的生命周期。如果对如何通过依赖注入框架来使用 MyBatis 感兴趣可以研究一下 MyBatis-Spring 或 MyBatis-Guice 两个子项目。

SqlSessionFactoryBuilder

这个类可以被实例化、使用和丢弃，一旦创建了 SqlSessionFactory，就不再需要它了。因此 SqlSessionFactoryBuilder 实例的最佳范围是方法范围（也就是局部方法变量）。你可以重用 SqlSessionFactoryBuilder 来创建多个 SqlSessionFactory 实例，但是最好还是不要让其一直存在以保证所有的 XML 解析资源开放给更重要的事情。

SqlSessionFactory

SqlSessionFactory 一旦被创建就应该在应用的运行期间一直存在，没有任何理由对它进行清除或重建。使用 SqlSessionFactory 的最佳实践是在应用运行期间不要重复创建多次，多次重建 SqlSessionFactory 被视为一种代码“坏味道（bad smell）”。因此 SqlSessionFactory 的最佳范围是应用范围。有很多方法可以做到，最简单的就是使用单例模式或者静态单例模式。

SqlSession

每个线程都应该有它自己的 SqlSession 实例。SqlSession 的实例不是线程安全的，因此是不能被共享的，所以它的最佳的范围是请求或方法范围。绝对不能将 SqlSession 实例的引用放在一个类的静态域，甚至一个类的实例变量也不行。也绝不能将 SqlSession 实例的引用放在任何类型的管理范围中，比如 Serlvet 架构中的 HttpSession。如果你现在正在使用一种 Web 框架，要考虑 SqlSession 放在一个和 HTTP 请求对象相似的范围中。换句话说，每次收到的 HTTP 请求，就可以打开一个 SqlSession，返回一个响应，就关闭它。这个关闭操作是很重要的，你应该把这个关闭操作放到 finally 块中以确保每次都能执行关闭。下面的示例就是一个确保 SqlSession 关闭的标准模式：

```
SqlSession session = sqlSessionFactory.openSession();
try {
  // do work
} finally {
  session.close();
}
```

在你的所有的代码中一致性地使用这种模式来保证所有数据库资源都能被正确地关闭。

映射器实例（Mapper Instances）

映射器是创建用来绑定映射语句的接口。映射器接口的实例是从 SqlSession 中获得的。因此从技术层面讲，映射器实例的最大范围是和 SqlSession 相同的，因为它们都是从 SqlSession 里被请求的。尽管如此，映射器实例的最佳范围是方法范围。也就是说，映射器实例应该在调用它们的方法中被请求，用过之后即可废弃。并不需要显式地关闭映射器实例，尽管在整个请求范围（request scope）保持映射器实例也不会有什么问题，但是很快你会发现，像 SqlSession 一样，在这个范围上管理太多的资源的话会难于控制。所以要保持简单，最好把映射器放在方法范围（method scope）内。下面的示例就展示了这个实践：

```
SqlSession session = sqlSessionFactory.openSession();
try {
  BlogMapper mapper = session.getMapper(BlogMapper.class);
  // do work
} finally {
  session.close();
}
```

XML 映射配置文件

MyBatis 的配置文件包含了影响 MyBatis 行为甚深的设置（settings）和属性（properties）信息。文档的顶层结构如下：

- configuration 配置
  - properties 属性
  - settings 设置
  - typeAliases 类型命名
  - typeHandlers 类型处理器
  - objectFactory 对象工厂
  - plugins 插件
  - environments 环境
    - environment 环境变量
      - transactionManager 事务管理器
      - dataSource 数据源
  - databaseIdProvider 数据库厂商标识
  - mappers 映射器

properties

这些属性都是可外部配置且可动态替换的，既可以在典型的 Java 属性文件中配置，亦可通过 properties 元素的子元素来传递。例如：

```
<properties resource="org/mybatis/example/config.properties">
  <property name="username" value="dev_user"/>
  <property name="password" value="F2Fa3!33TYyg"/>
</properties>
```

其中的属性就可以在整个配置文件中使用来替换需要动态配置的属性值。比如:

```
<dataSource type="POOLED">
  <property name="driver" value="${driver}"/>
  <property name="url" value="${url}"/>
  <property name="username" value="${username}"/>
  <property name="password" value="${password}"/>
</dataSource>
```

这个例子中的 username 和 password 将会由 properties 元素中设置的相应值来替换。 driver 和 url 属性将会由 config.properties 文件中对应的值来替换。这样就为配置提供了诸多灵活选择。

属性也可以被传递到 SqlSessionBuilder.build()方法中。例如：

```
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,

// ... or ...

SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,
```

如果属性在不只一个地方进行了配置，那么 MyBatis 将按照下面的顺序来加载：

- 在 properties 元素体内指定的属性首先被读取。
- 然后根据 properties 元素中的 resource 属性读取类路径下属性文件或根据 url 属性指定的路径读取属性文件，并覆盖已读取的同名属性。
- 最后读取作为方法参数传递的属性，并覆盖已读取的同名属性。

因此，通过方法参数传递的属性具有最高优先级，resource/url 属性中指定的配置文件次之，最低优先级的是 properties 属性中指定的属性。

settings

这是 MyBatis 中极为重要的调整设置，它们会改变 MyBatis 的运行时行为。下表描述了设置中各项的意图、默认值等。

设置参数	描述	有效值
cacheEnabled	该配置影响的所有映射器中配置的缓存的全局开关。	true | false
lazyLoadingEnabled	延迟加载的全局开关。当开启时，所有关联对象都会延迟加载。 特定关联关系中可通过设置 `fetchType` 属性来覆盖该项的开关状态。	true | false
aggressiveLazyLoading	当启用时，对任意延迟属性的调用会使带有延迟加载属性的对象完整加载；反之，每种属性将会按需加载。	true | false
multipleResultSetsEnabled	是否允许单一语句返回多结果集（需要兼容驱动）。	true | false
	使用列标签代替列	

useColumnLabel	名。不同的驱动在这方面会有不同的表现， 具体可参考相关驱动文档或通过测试这两种不同的模式来观察所用驱动的结果。	true | false
useGeneratedKeys	允许 JDBC 支持自动生成主键，需要驱动兼容。 如果设置为 true 则这个设置强制使用自动生成主键，尽管一些驱动不能兼容但仍可正常工作（比如 Derby）。	true | false
autoMappingBehavior	指定 MyBatis 应如何自动映射列到字段或属性。 NONE 表示取消自动映射；PARTIAL 只会自动映射没有定义嵌套结果集映射的结果集。 FULL 会自动映射任意复杂的结果集（无论是否嵌套）。	NONE, PARTIAL, FULL
defaultExecutorType	配置默认的执行器。SIMPLE 就是普通的执行器；REUSE 执行器会重用预处理语句（prepared statements）；BATCH 执行器将重用语句并执行批量更新。	SIMPLE REUSE BATCH
defaultStatementTimeout	设置超时时间，它决定驱动等待数据库响应的秒数。	Any positive integer
defaultFetchSize	Sets the driver a hint as to control fetching size for return results. This parameter value	Any positive integer

	can be override by a query setting.	
safeRowBoundsEnabled	允许在嵌套语句中使用分页（RowBounds）。	true | false
mapUnderscoreToCamelCase	是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 A_COLUMN 到经典 Java 属性名 aColumn 的类似映射。	true | false
localCacheScope	MyBatis 利用本地缓存机制（Local Cache）防止循环引用（circular references）和加速重复嵌套查询。默认值为 SESSION，这种情况下会缓存一个会话中执行的所有查询。 若设置值为 STATEMENT，本地会话仅用在语句执行上，对相同 SqlSession 的不同调用将不会共享数据。	SESSION | STATEMENT
jdbcTypeForNull	当没有为参数提供特定的 JDBC 类型时，为空值指定 JDBC 类型。 某些驱动需要指定列的 JDBC 类型，多数情况直接用一般类型即可，比如 NULL、VARCHAR 或 OTHER。	JdbcType enumeration. Most common are: NULL, VARCHAR and OTHER
lazyLoadTriggerMethods	指定哪个对象的方法触发一次延迟加载。	A method name list separated by commas
defaultScriptingLanguage	指定动态 SQL 生	A type alias or fully

defaultScriptingLanguage	成的默认语言。	qualified class name.
callSettersOnNulls	指定当结果集中值为 null 的时候是否调用映射对象的 setter（map 对象时为 put）方法，这对于有 Map.keySet() 依赖或 null 值初始化的时候是有用的。注意基本类型（int、boolean等）是不能设置成 null 的。	true | false
logPrefix	指定 MyBatis 增加到日志名称的前缀。	Any String
logImpl	指定 MyBatis 所用日志的具体实现，未指定时将自动查找。	SLF4J | LOG4J | LOG4J2 | JDK_LOGGING | COMMONS_LOGGING | STDOUT_LOGGING | NO_LOGGING
proxyFactory	指定 Mybatis 创建具有延迟加载能力的对象所用到的代理工具。	CGLIB | JAVASSIST
vfsImpl	Specifies VFS implementations	Fully qualified class names of custom VFS implementation separated by commas.

一个配置完整的 settings 元素的示例如下：

```
<settings>
  <setting name="cacheEnabled" value="true"/>
  <setting name="lazyLoadingEnabled" value="true"/>
  <setting name="multipleResultSetsEnabled" value="true"/>
  <setting name="useColumnLabel" value="true"/>
  <setting name="useGeneratedKeys" value="false"/>
  <setting name="autoMappingBehavior" value="PARTIAL"/>
  <setting name="defaultExecutorType" value="SIMPLE"/>
  <setting name="defaultStatementTimeout" value="25"/>
  <setting name="defaultFetchSize" value="100"/>
  <setting name="safeRowBoundsEnabled" value="false"/>
  <setting name="mapUnderscoreToCamelCase" value="false"/>
  <setting name="localCacheScope" value="SESSION"/>
  <setting name="jdbcTypeForNull" value="OTHER"/>
  <setting name="lazyLoadTriggerMethods" value="equals,clone,hashCo
</settings>
```

typeAliases

类型别名是为 Java 类型设置一个短的名字。它只和 XML 配置有关，存在的意义仅在于用来减少类完全限定名的冗余。例如:

```
<typeAliases>
  <typeAlias alias="Author" type="domain.blog.Author"/>
  <typeAlias alias="Blog" type="domain.blog.Blog"/>
  <typeAlias alias="Comment" type="domain.blog.Comment"/>
  <typeAlias alias="Post" type="domain.blog.Post"/>
  <typeAlias alias="Section" type="domain.blog.Section"/>
  <typeAlias alias="Tag" type="domain.blog.Tag"/>
</typeAliases>
```

当这样配置时， `Blog` 可以用在任何使用 `domain.blog.Blog` 的地方。

也可以指定一个包名，MyBatis 会在包名下面搜索需要的 Java Bean，比如:

```
<typeAliases>
  <package name="domain.blog"/>
</typeAliases>
```

每一个在包 `domain.blog` 中的 Java Bean，在没有注解的情况下，会使用 Bean 的首字母小写的非限定类名来作为它的别名。 比如 `domain.blog.Author` 的别名为 `author` ；若有注解，则别名为其注解值。看下面的例子：

```
@Alias("author")
public class Author {
    ...
}
```

已经为许多常见的 Java 类型内建了相应的类型别名。它们都是大小写不敏感的，需要注意的是由基本类型名称重复导致的特殊处理。

别名	映射的类型
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer
double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator

typeHandlers

无论是 MyBatis 在预处理语句（PreparedStatement）中设置一个参数时，还是从结果集中取出一个值时， 都会用类型处理器将获取的值以合适的方式转换成 Java 类型。下表描述了一些默认的类型处理器。

类型处理器	Java 类型	JDBC 类型
`BooleanTypeHandler`	`java.lang.Boolean` , `boolean`	数据库兼容的 `BOOLEAN`
`ByteTypeHandler`	`java.lang.Byte` , `byte`	数据库兼容的 `NUMERIC` 或 `BYTE`
`ShortTypeHandler`	`java.lang.Short` , `short`	数据库兼容的 `NUMERIC` 或 `SHORT INTEGER`
`IntegerTypeHandler`	`java.lang.Integer` , `int`	数据库兼容的 `NUMERIC` 或 `INTEGER`
`LongTypeHandler`	`java.lang.Long` , `long`	数据库兼容的 `NUMERIC` 或 `LONG INTEGER`
`FloatTypeHandler`	`java.lang.Float` , `float`	数据库兼容的 `NUMERIC` 或 `FLOAT`
`DoubleTypeHandler`	`java.lang.Double` , `double`	数据库兼容的 `NUMERIC` 或 `DOUBLE`
`BigDecimalTypeHandler`	`java.math.BigDecimal`	数据库兼容的 `NUMERIC` 或 `DECIMAL`
`StringTypeHandler`	`java.lang.String`	`CHAR` , `VARCHAR`
`ClobTypeHandler`	`java.lang.String`	`CLOB` , `LONGVARCHAR`
`NStringTypeHandler`	`java.lang.String`	`NVARCHAR` , `NCHAR`
`NClobTypeHandler`	`java.lang.String`	`NCLOB`
`ByteArrayTypeHandler`	`byte[]`	数据库兼容的字节流类型
`BlobTypeHandler`	`byte[]`	`BLOB` , `LONGVARBINARY`

`DateTypeHandler`	`java.util.Date`	`TIMESTAMP`
`DateOnlyTypeHandler`	`java.util.Date`	`DATE`
`TimeOnlyTypeHandler`	`java.util.Date`	`TIME`
`SqlTimestampTypeHandler`	`java.sql.Timestamp`	`TIMESTAMP`
`SqlDateTypeHandler`	`java.sql.Date`	`DATE`
`SqlTimeTypeHandler`	`java.sql.Time`	`TIME`
`ObjectTypeHandler`	Any	`OTHER` 或未指定类型
`EnumTypeHandler`	Enumeration Type	VARCHAR-任何兼容的字符串类型，存储枚举的名称（而不是索引）
`EnumOrdinalTypeHandler`	Enumeration Type	任何兼容的 `NUMERIC` 或 `DOUBLE` 类型，存储枚举的索引（而不是名称）。

你可以重写类型处理器或创建你自己的类型处理器来处理不支持的或非标准的类型。 具体做法为：实现 `org.apache.ibatis.type.TypeHandler` 接口， 或继承一个很便利的类 `org.apache.ibatis.type.BaseTypeHandler` ， 然后可以选择性地将它映射到一个 JDBC 类型。比如：

```
// ExampleTypeHandler.java
@MappedJdbcTypes(JdbcType.VARCHAR)
public class ExampleTypeHandler extends BaseTypeHandler<String> {

  @Override
  public void setNonNullParameter(PreparedStatement ps, int i, Stri
    ps.setString(i, parameter);
  }

  @Override
  public String getNullableResult(ResultSet rs, String columnName)
    return rs.getString(columnName);
  }

  @Override
  public String getNullableResult(ResultSet rs, int columnIndex) th
    return rs.getString(columnIndex);
  }

  @Override
  public String getNullableResult(CallableStatement cs, int columnI
    return cs.getString(columnIndex);
  }
}
```

```
<!-- mybatis-config.xml -->
<typeHandlers>
  <typeHandler handler="org.mybatis.example.ExampleTypeHandler"/>
</typeHandlers>
```

使用这个的类型处理器将会覆盖已经存在的处理 Java 的 String 类型属性和 VARCHAR 参数及结果的类型处理器。 要注意 MyBatis 不会窥探数据库元信息来决定使用哪种类型，所以你必须在参数和结果映射中指明那是 VARCHAR 类型的字段， 以使其能够绑定到正确的类型处理器上。 这是因为：MyBatis 直到语句被执行才清楚数据类型。

通过类型处理器的泛型，MyBatis 可以得知该类型处理器处理的 Java 类型，不过这种行为可以通过两种方法改变：

- 在类型处理器的配置元素（typeHandler element）上增加一个 `javaType` 属性（比如：`javaType="String"`）；
- 在类型处理器的类上（TypeHandler class）增加一个 `@MappedTypes` 注解来指定与其关联的 Java 类型列表。 如果在 `javaType` 属性中也同时指定，则注解方式将被忽略。

可以通过两种方式来指定被关联的 JDBC 类型：

- 在类型处理器的配置元素上增加一个 `javaType` 属性（比如： `javaType="VARCHAR"` ）；
- 在类型处理器的类上（TypeHandler class）增加一个 `@MappedJdbcTypes` 注解来指定与其关联的 JDBC 类型列表。 如果在 `javaType` 属性中也同时指定，则注解方式将被忽略。

最后，可以让 MyBatis 为你查找类型处理器：

```
<!-- mybatis-config.xml -->
<typeHandlers>
  <package name="org.mybatis.example"/>
</typeHandlers>
```

注意在使用自动检索（autodiscovery）功能的时候，只能通过注解方式来指定 JDBC 的类型。

你能创建一个泛型类型处理器，它可以处理多于一个类。为达到此目的， 需要增加一个接收该类作为参数的构造器，这样在构造一个类型处理器的时候 MyBatis 就会传入一个具体的类。

```
//GenericTypeHandler.java
public class GenericTypeHandler<E extends MyObject> extends BaseTyp

  private Class<E> type;

  public GenericTypeHandler(Class<E> type) {
    if (type == null) throw new IllegalArgumentException("Type argu
    this.type = type;
  }
  ...
```

`EnumTypeHandler` 和 `EnumOrdinalTypeHandler` 都是泛型类型处理器（generic TypeHandlers）， 我们将会在接下来的部分详细探讨。

处理枚举类型

若想映射枚举类型 `Enum` ，则需要从 `EnumTypeHandler` 或者 `EnumOrdinalTypeHandler` 中选一个来使用。

比如说我们想存储取近似值时用到的舍入模式。默认情况下，MyBatis 会利用 `EnumTypeHandler` 来把 `Enum` 值转换成对应的名字。

注意 **`EnumTypeHandler`** 在某种意义上来说是比较特别的，其他的处理器只针对某个特定的类，而它不同，它会处理任意继承了 **`Enum`** 的类。

不过，我们可能不想存储名字，相反我们的 DBA 会坚持使用整形值代码。那也一样轻而易举： 在配置文件中把 `EnumOrdinalTypeHandler` 加到 `typeHandlers` 中即可， 这样每个 `RoundingMode` 将通过他们的序数值来映射成对应的整形。

```
<!-- mybatis-config.xml -->
<typeHandlers>
  <typeHandler handler="org.apache.ibatis.type.EnumOrdinalTypeHandl
</typeHandlers>
```

但是怎样能将同样的 `Enum` 既映射成字符串又映射成整形呢？

自动映射器（auto-mapper）会自动地选用 `EnumOrdinalTypeHandler` 来处理，所以如果我们想用普通的 `EnumTypeHandler` ，就非要为那些 SQL 语句显式地设置要用到的类型处理器不可。

（下一节才开始讲映射器文件，所以如果是首次阅读该文档，你可能需要先越过这一步，过会再来看。）

```
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="org.apache.ibatis.submitted.rounding.Mapper">
    <resultMap type="org.apache.ibatis.submitted.rounding.User" id=
        <id column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="funkyNumber" property="funkyNumber"/>
        <result column="roundingMode" property="roundingMode"/>
    </resultMap>

    <select id="getUser" resultMap="usermap">
        select * from users
    </select>
    <insert id="insert">
        insert into users (id, name, funkyNumber, roundingMode) val
            #{id}, #{name}, #{funkyNumber}, #{roundingMode}
        )
    </insert>

    <resultMap type="org.apache.ibatis.submitted.rounding.User" id=
        <id column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="funkyNumber" property="funkyNumber"/>
        <result column="roundingMode" property="roundingMode" typeH
    </resultMap>
    <select id="getUser2" resultMap="usermap2">
        select * from users2
    </select>
    <insert id="insert2">
        insert into users2 (id, name, funkyNumber, roundingMode) va
            #{id}, #{name}, #{funkyNumber}, #{roundingMode, typeHan
        )
    </insert>

</mapper>
```

注意，这里的 select 语句强制使用 `resultMap` 来代替 `resultType` 。

对象工厂（objectFactory）

MyBatis 每次创建结果对象的新实例时，它都会使用一个对象工厂（ObjectFactory）实例来完成。 默认的对象工厂需要做的仅仅是实例化目标类，要么通过默认构造方法，要么在参数映射存在的时候通过参数构造方法来实例化。 如果想覆盖对象工厂的默认行为，则可以通过创建自己的对象工厂来实现。比如：

```
// ExampleObjectFactory.java
public class ExampleObjectFactory extends DefaultObjectFactory {
  public Object create(Class type) {
    return super.create(type);
  }
  public Object create(Class type, List<Class> constructorArgTypes,
    return super.create(type, constructorArgTypes, constructorArgs)
  }
  public void setProperties(Properties properties) {
    super.setProperties(properties);
  }
  public <T> boolean isCollection(Class<T> type) {
    return Collection.class.isAssignableFrom(type);
  }}
```

```
<!-- mybatis-config.xml -->
<objectFactory type="org.mybatis.example.ExampleObjectFactory">
  <property name="someProperty" value="100"/>
</objectFactory>
```

ObjectFactory 接口很简单，它包含两个创建用的方法，一个是处理默认构造方法的，另外一个是处理带参数的构造方法的。 最后，setProperties 方法可以被用来配置 ObjectFactory，在初始化你的 ObjectFactory 实例后， objectFactory 元素体中定义的属性会被传递给 setProperties 方法。

插件（**plugins**）

MyBatis 允许你在已映射语句执行过程中的某一点进行拦截调用。默认情况下，MyBatis 允许使用插件来拦截的方法调用包括：

- Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
- ParameterHandler (getParameterObject, setParameters)
- ResultSetHandler (handleResultSets, handleOutputParameters)
- StatementHandler (prepare, parameterize, batch, update, query)

这些类中方法的细节可以通过查看每个方法的签名来发现，或者直接查看 MyBatis 的发行包中的源代码。 假设你想做的不仅仅是监控方法的调用，那么你应该很好的了解正在重写的方法的行为。 因为如果在试图修改或重写已有方法的行为的时候，你很可能在破坏 MyBatis 的核心模块。 这些都是更低层的类和方法，所以使用插件的时候要特别当心。

通过 MyBatis 提供的强大机制，使用插件是非常简单的，只需实现 Interceptor 接口，并指定了想要拦截的方法签名即可。

```
// ExamplePlugin.java
@Intercepts({@Signature(
  type= Executor.class,
  method = "update",
  args = {MappedStatement.class,Object.class})})
public class ExamplePlugin implements Interceptor {
  public Object intercept(Invocation invocation) throws Throwable
    return invocation.proceed();
  }
  public Object plugin(Object target) {
    return Plugin.wrap(target, this);
  }
  public void setProperties(Properties properties) {
  }
}
```

```
<!-- mybatis-config.xml -->
<plugins>
  <plugin interceptor="org.mybatis.example.ExamplePlugin">
    <property name="someProperty" value="100"/>
  </plugin>
</plugins>
```

上面的插件将会拦截在 Executor 实例中所有的 “update” 方法调用， 这里的 Executor 是负责执行低层映射语句的内部对象。

NOTE 覆盖配置类

除了用插件来修改 MyBatis 核心行为之外，还可以通过完全覆盖配置类来达到目的。只需继承后覆盖其中的每个方法，再把它传递到 sqlSessionFactoryBuilder.build(myConfig) 方法即可。再次重申，这可能会严重影响 MyBatis 的行为，务请慎之又慎。

配置环境（environments）

MyBatis 可以配置成适应多种环境，这种机制有助于将 SQL 映射应用于多种数据库之中， 现实情况下有多种理由需要这么做。例如，开发、测试和生产环境需要有不同的配置；或者共享相同 Schema 的多个生产数据库， 想使用相同的 SQL 映射。许多类似的用例。

不过要记住：尽管可以配置多个环境，每个 **SqlSessionFactory** 实例只能选择其一。

所以，如果你想连接两个数据库，就需要创建两个 SqlSessionFactory 实例，每个数据库对应一个。而如果是三个数据库，就需要三个实例，依此类推，记起来很简单：

- 每个数据库对应一个 **SqlSessionFactory** 实例

为了指定创建哪种环境，只要将它作为可选的参数传递给 SqlSessionFactoryBuilder 即可。可以接受环境配置的两个方法签名是：

```
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,
```

如果忽略了环境参数，那么默认环境将会被加载，如下所示：

```
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader);
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,p
```

环境元素定义了如何配置环境。

```
<environments default="development">
  <environment id="development">
    <transactionManager type="JDBC">
      <property name="..." value="..."/>
    </transactionManager>
    <dataSource type="POOLED">
      <property name="driver" value="${driver}"/>
      <property name="url" value="${url}"/>
      <property name="username" value="${username}"/>
      <property name="password" value="${password}"/>
    </dataSource>
  </environment>
</environments>
```

注意这里的关键点:

- 默认的环境 ID（比如:default="development"）。
- 每个 environment 元素定义的环境 ID（比如:id="development"）。
- 事务管理器的配置（比如:type="JDBC"）。
- 数据源的配置（比如:type="POOLED"）。

默认的环境和环境 ID 是一目了然的。随你怎么命名，只要保证默认环境要匹配其中一个环境ID。

事务管理器（**transactionManager**）

在 MyBatis 中有两种类型的事务管理器（也就是 type="[JDBC|MANAGED]"）：

- JDBC – 这个配置就是直接使用了 JDBC 的提交和回滚设置，它依赖于从数据源得到的连接来管理事务范围。

- MANAGED – 这个配置几乎没做什么。它从来不提交或回滚一个连接，而是让容器来管理事务的整个生命周期（比如 JEE 应用服务器的上下文）。 默认情况下它会关闭连接，然而一些容器并不希望这样，因此需要将 closeConnection 属性设置为 false 来阻止它默认的关闭行为。例如:

```
&lt;transactionManager type="MANAGED"&gt;
  &lt;property name="closeConnection" value="false"/&gt;
&lt;/transactionManager&gt;
```

NOTE如果你正在使用 Spring + MyBatis，则没有必要配置事务管理器， 因为 Spring 模块会使用自带的管理器来覆盖前面的配置。

这两种事务管理器类型都不需要任何属性。它们不过是类型别名，换句话说，你可以使用 TransactionFactory 接口的实现类的完全限定名或类型别名代替它们。

```
public interface TransactionFactory {
  void setProperties(Properties props);
  Transaction newTransaction(Connection conn);
  Transaction newTransaction(DataSource dataSource, TransactionIsol
}
```

任何在 XML 中配置的属性在实例化之后将会被传递给 setProperties() 方法。你也需要创建一个 Transaction 接口的实现类，这个接口也很简单：

```
public interface Transaction {
  Connection getConnection() throws SQLException;
  void commit() throws SQLException;
  void rollback() throws SQLException;
  void close() throws SQLException;
}
```

使用这两个接口，你可以完全自定义 MyBatis 对事务的处理。

数据源（**dataSource**）

dataSource 元素使用标准的 JDBC 数据源接口来配置 JDBC 连接对象的资源。

- 许多 MyBatis 的应用程序将会按示例中的例子来配置数据源。然而它并不是必须的。要知道为了方便使用延迟加载，数据源才是必须的。

有三种内建的数据源类型（也就是 type="[UNPOOLED|POOLED|JNDI]"）：

UNPOOLED– 这个数据源的实现只是每次被请求时打开和关闭连接。虽然一点慢，它对在及时可用连接方面没有性能要求的简单应用程序是一个很好的选择。 不同的数据库在这方面表现也是不一样的，所以对某些数据库来说使用连接池并不重要，这个配置也是理想的。UNPOOLED 类型的数据源仅仅需要配置以下 5 种属性：

- `driver` – 这是 JDBC 驱动的 Java 类的完全限定名（并不是JDBC驱动中可能包含的数据源类）。
- `url` – 这是数据库的 JDBC URL 地址。
- `username` – 登录数据库的用户名。
- `password` – 登录数据库的密码。
- `defaultTransactionIsolationLevel` – 默认的连接事务隔离级别。

作为可选项，你也可以传递属性给数据库驱动。要这样做，属性的前缀为“driver.”，例如：

- `driver.encoding=UTF8`

这将通过DriverManager.getConnection(url,driverProperties)方法传递值为 `UTF8` 的 `encoding` 属性给数据库驱动。

POOLED– 这种数据源的实现利用“池”的概念将 JDBC 连接对象组织起来，避免了创建新的连接实例时所必需的初始化和认证时间。 这是一种使得并发 Web 应用快速响应请求的流行处理方式。

除了上述提到 UNPOOLED 下的属性外，会有更多属性用来配置 POOLED 的数据源：

- `poolMaximumActiveConnections` – 在任意时间可以存在的活动（也就是正在使用）连接数量，默认值：10
- `poolMaximumIdleConnections` – 任意时间可能存在的空闲连接数。
- `poolMaximumCheckoutTime` – 在被强制返回之前，池中连接被检出（checked out）时间，默认值：20000 毫秒（即 20 秒）
- `poolTimeToWait` – 这是一个底层设置，如果获取连接花费的相当长的时间，它会给连接池打印状态日志并重新尝试获取一个连接（避免在误配置的情况下一直安静的失败），默认值：20000 毫秒（即 20 秒）。
- `poolPingQuery` – 发送到数据库的侦测查询，用来检验连接是否处在正常工作秩序中并准备接受请求。默认是“NO PING QUERY SET”，这会导致多数数据库驱动失败时带有一个恰当的错误消息。
- `poolPingEnabled` – 是否启用侦测查询。若开启，也必须使用一个可执行的 SQL 语句设置 `poolPingQuery` 属性（最好是一个非常快的 SQL），默认值：false。
- `poolPingConnectionsNotUsedFor` – 配置 poolPingQuery 的使用频度。这可以被设置成匹配具体的数据库连接超时时间，来避免不必要的侦测，默认值：0（即所有连接每一时刻都被侦测 — 当然仅当 poolPingEnabled 为 true 时适用）。

JNDI– 这个数据源的实现是为了能在如 EJB 或应用服务器这类容器中使用，容器可以集中或在外部配置数据源，然后放置一个 JNDI 上下文的引用。这种数据源配置只需要两个属性：

- `initial_context` – 这个属性用来在 InitialContext 中寻找上下文（即，initialContext.lookup(initial_context)）。这是个可选属性，如果忽略，那么 data_source 属性将会直接从 InitialContext 中寻找。
- `data_source` – 这是引用数据源实例位置的上下文的路径。提供了

initial_context 配置时会在其返回的上下文中进行查找，没有提供时则直接在 InitialContext 中查找。

和其他数据源配置类似，可以通过添加前缀“env.”直接把属性传递给初始上下文。比如：

- `env.encoding=UTF8`

这就会在初始上下文（InitialContext）实例化时往它的构造方法传递值为 `UTF8` 的 `encoding` 属性。

通过需要实现接口 `org.apache.ibatis.datasource.DataSourceFactory` ，也可使用任何第三方数据源，：

```
public interface DataSourceFactory {
  void setProperties(Properties props);
  DataSource getDataSource();
}
```

`org.apache.ibatis.datasource.unpooled.UnpooledDataSourceFactory` 可被用作父类来构建新的数据源适配器，比如下面这段插入 C3P0 数据源所必需的代码：

```
import org.apache.ibatis.datasource.unpooled.UnpooledDataSourceFact
import com.mchange.v2.c3p0.ComboPooledDataSource;

public class C3P0DataSourceFactory extends UnpooledDataSourceFacto

  public C3P0DataSourceFactory() {
    this.dataSource = new ComboPooledDataSource();
  }
}
```

为了令其工作，为每个需要 MyBatis 调用的 setter 方法中增加一个属性。下面是一个可以连接至 PostgreSQL 数据库的例子：

```
<dataSource type="org.myproject.C3P0DataSourceFactory">
  <property name="driver" value="org.postgresql.Driver"/>
  <property name="url" value="jdbc:postgresql:mydb"/>
  <property name="username" value="postgres"/>
  <property name="password" value="root"/>
</dataSource>
```

databaseIdProvider

MyBatis 可以根据不同的数据库厂商执行不同的语句，这种多厂商的支持是基于映射语句中的 `databaseId` 属性。MyBatis 会加载不带 `databaseId` 属性和带有匹配当前数据库 `databaseId` 属性的所有语句。如果同时找到带有 `databaseId` 和不带 `databaseId` 的相同语句，则后者会被舍弃。为支持多厂商特性只要像下面这样在 mybatis-config.xml 文件中加入 `databaseIdProvider` 即可：

```
<databaseIdProvider type="DB_VENDOR" />
```

这里的 DB_VENDOR 会通过 `DatabaseMetaData#getDatabaseProductName()` 返回的字符串进行设置。由于通常情况下这个字符串都非常长而且相同产品的不同版本会返回不同的值，所以最好通过设置属性别名来使其变短，如下：

```
<databaseIdProvider type="DB_VENDOR">
  <property name="SQL Server" value="sqlserver"/>
  <property name="DB2" value="db2"/>
  <property name="Oracle" value="oracle" />
</databaseIdProvider>
```

在有 properties 时，DB_VENDOR databaseIdProvider 的将被设置为第一个能匹配数据库产品名称的属性键对应的值，如果没有匹配的属性将会设置为 “null”。在这个例子中，如果 `getDatabaseProductName()` 返回“Oracle (DataDirect)”，databaseId 将被设置为“oracle”。

你可以通过实现接口 `org.apache.ibatis.mapping.DatabaseIdProvider` 并在 mybatis-config.xml 中注册来构建自己的 DatabaseIdProvider：

```
public interface DatabaseIdProvider {
  void setProperties(Properties p);
  String getDatabaseId(DataSource dataSource) throws SQLException;
}
```

映射器（mappers）

既然 MyBatis 的行为已经由上述元素配置完了，我们现在就要定义 SQL 映射语句了。但是首先我们需要告诉 MyBatis 到哪里去找到这些语句。Java 在自动查找这方面没有提供一个很好的方法，所以最佳的方式是告诉 MyBatis 到哪里去找映射文件。你可以使用相对于类路径的资源引用，或完全限定资源定位符（包括 `file:///` 的 URL），或类名和包名等。例如：

```
<!-- Using classpath relative resources -->
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>
```

```
<!-- Using url fully qualified paths -->
<mappers>
  <mapper url="file:///var/mappers/AuthorMapper.xml"/>
  <mapper url="file:///var/mappers/BlogMapper.xml"/>
  <mapper url="file:///var/mappers/PostMapper.xml"/>
</mappers>
```

```
<!-- Using mapper interface classes -->
<mappers>
  <mapper class="org.mybatis.builder.AuthorMapper"/>
  <mapper class="org.mybatis.builder.BlogMapper"/>
  <mapper class="org.mybatis.builder.PostMapper"/>
</mappers>
```

```
<!-- Register all interfaces in a package as mappers -->
<mappers>
  <package name="org.mybatis.builder"/>
</mappers>
```

这些配置会告诉了 MyBatis 去哪里找映射文件，剩下的细节就应该是每个 SQL 映射文件了，也就是接下来我们要讨论的。

Mapper XML 文件

MyBatis 的真正强大在于它的映射语句，也是它的魔力所在。由于它的异常强大，映射器的 XML 文件就显得相对简单。如果拿它跟具有相同功能的 JDBC 代码进行对比，你会立即发现省掉了将近 95% 的代码。MyBatis 就是针对 SQL 构建的，并且比普通的方法做的更好。

SQL 映射文件有很少的几个顶级元素（按照它们应该被定义的顺序）：

- `cache` – 给定命名空间的缓存配置。
- `cache-ref` – 其他命名空间缓存配置的引用。
- `resultMap` – 是最复杂也是最强大的元素，用来描述如何从数据库结果集中来加载对象。
- `~~parameterMap~~` ~~– 已废弃！老式风格的参数映射。内联参数是首选,这个元素可能在将来被移除，这里不会记录。~~
- `sql` – 可被其他语句引用的可重用语句块。
- `insert` – 映射插入语句
- `update` – 映射更新语句
- `delete` – 映射删除语句
- `select` – 映射查询语句

下一部分将从语句本身开始来描述每个元素的细节。

select

查询语句是 MyBatis 中最常用的元素之一，光能把数据存到数据库中价值并不大，如果还能重新取出来才有用，多数应用也都是查询比修改要频繁。对每个插入、更新或删除操作，通常对应多个查询操作。这是 MyBatis 的基本原则之一，也是将焦点和努力放到查询和结果映射的原因。简单查询的 select 元素是非常简单的。比如：

```
<select id="selectPerson" parameterType="int" resultType="hashmap">
  SELECT * FROM PERSON WHERE ID = #{id}
</select>
```

这个语句被称作 selectPerson，接受一个 int（或 Integer）类型的参数，并返回一个 HashMap 类型的对象，其中的键是列名，值便是结果行中的对应值。

注意参数符号：

```
#{id}
```

这就告诉 MyBatis 创建一个预处理语句参数，通过 JDBC，这样的一个参数在 SQL 中会由一个“?”来标识，并被传递到一个新的预处理语句中，就像这样：

```
// Similar JDBC code, NOT MyBatis…
String selectPerson = "SELECT * FROM PERSON WHERE ID=?";
PreparedStatement ps = conn.prepareStatement(selectPerson);
ps.setInt(1,id);
```

当然，这需要很多单独的 JDBC 的代码来提取结果并将它们映射到对象实例中，这就是 MyBatis 节省你时间的地方。我们需要深入了解参数和结果映射，细节部分我们下面来了解。

select 元素有很多属性允许你配置，来决定每条语句的作用细节。

```
<select
  id="selectPerson"
  parameterType="int"
  parameterMap="deprecated"
  resultType="hashmap"
  resultMap="personResultMap"
  flushCache="false"
  useCache="true"
  timeout="10000"
  fetchSize="256"
  statementType="PREPARED"
  resultSetType="FORWARD_ONLY">
```

Select Attributes

属性	描述
`id`	在命名空间中唯一的标识符，可以被用来引用这条语句。
`parameterType`	将会传入这条语句的参数类的完全限定名或别名。这个属性是可选的，因为 MyBatis 可以通过 TypeHandler 推断出具体传入语句的参数，默认值为 unset。
~~parameterMap~~	~~这是引用外部 parameterMap 的已经被废弃的方法。使用内联参数映射和 parameterType 属性。~~
`resultType`	从这条语句中返回的期望类型的类的完全限定名或别名。注意如果是集合情形，那应该是集合可以包含的类型，而不能是集合本身。使用 resultType 或 resultMap，但不能同时使用。
`resultMap`	外部 resultMap 的命名引用。结果集的映射是 MyBatis 最强大的特性，对其有一个很好的理解的话，许多复杂映射的情形都能迎刃而解。使用 resultMap 或 resultType，但不能同时使用。

`flushCache`	将其设置为 true，任何时候只要语句被调用，都会导致本地缓存和二级缓存都会被清空，默认值：false。
`useCache`	将其设置为 true，将会导致本条语句的结果被二级缓存，默认值：对 select 元素为 true。
`timeout`	这个设置是在抛出异常之前，驱动程序等待数据库返回请求结果的秒数。默认值为 unset（依赖驱动）。
`fetchSize`	这是尝试影响驱动程序每次批量返回的结果行数和这个设置值相等。默认值为 unset（依赖驱动）。
`statementType`	STATEMENT，PREPARED 或 CALLABLE 的一个。这会让 MyBatis 分别使用 Statement，PreparedStatement 或 CallableStatement，默认值：PREPARED。
`resultSetType`	FORWARD_ONLY，SCROLL_SENSITIVE 或 SCROLL_INSENSITIVE 中的一个，默认值为 unset （依赖驱动）。
`databaseId`	如果配置了 databaseIdProvider，MyBatis 会加载所有的不带 databaseId 或匹配当前 databaseId 的语句；如果带或者不带的语句都有，则不带的会被忽略。
`resultOrdered`	这个设置仅针对嵌套结果 select 语句适用：如果为 true，就是假设包含了嵌套结果集或是分组了，这样的话当返回一个主结果行的时候，就不会发生有对前面结果集的引用的情况。这就使得在获取嵌套的结果集的时候不至于导致内存不够用。默认值： `false` 。
`resultSets`	这个设置仅对多结果集的情况适用，它将列出语句执行后返回的结果集并每个结果集给一个名称，名称是逗号分隔的。

insert, update 和 delete

数据变更语句 insert，update 和 delete 的实现非常接近：

```
<insert
  id="insertAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  keyProperty=""
  keyColumn=""
  useGeneratedKeys=""
  timeout="20">

<update
  id="updateAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  timeout="20">

<delete
  id="deleteAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  timeout="20">
```

Insert, Update 和 Delete 的属性

属性	描述
`id`	命名空间中的唯一标识符，可被用来代表这条语句。
`parameterType`	将要传入语句的参数的完全限定类名或别名。这个属性是可选的，因为 MyBatis 可以通过 TypeHandler 推断出具体传入语句的参数，默认值为 unset。
`~~parameterMap~~`	~~这是引用外部 parameterMap 的已经被废弃的方法。使用内联参数映射和 parameterType 属性。~~
`flushCache`	将其设置为 true，任何时候只要语句被调用，都会导致本地缓存和二级缓存都会被清空，默认值：true（对应插入、更新和删除语句）。
`timeout`	这个设置是在抛出异常之前，驱动程序等待数据库返回请求结果的秒数。默认值为 unset（依赖驱动）。
`statementType`	STATEMENT，PREPARED 或 CALLABLE 的一个。这会让 MyBatis 分别使用 Statement，PreparedStatement 或 CallableStatement，默认值：PREPARED。
`useGeneratedKeys`	（仅对 insert 和 update 有用）这会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键（比如：像 MySQL 和 SQL Server 这样的关系数据库管理系统的自动递增字段），默认值：false。
`keyProperty`	（仅对 insert 和 update 有用）唯一标记一个属性，MyBatis 会通过 getGeneratedKeys 的返回值或者通过 insert 语句的 selectKey 子元素设置它的键值，默认：`unset`。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
`keyColumn`	（仅对 insert 和 update 有用）通过生成的键值设置表中的列名，这个设置仅在某些数据库（像 PostgreSQL）是必须的，当主键列不是表中的第一列的时候需要设置。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
`databaseId`	如果配置了 databaseIdProvider，MyBatis 会加载所有的不带 databaseId 或匹配当前 databaseId 的语句；如果带或者不带的语句都有，则不带的会被忽略。

下面就是 insert，update 和 delete 语句的示例：

```
<insert id="insertAuthor">
  insert into Author (id,username,password,email,bio)
  values (#{id},#{username},#{password},#{email},#{bio})
</insert>

<update id="updateAuthor">
  update Author set
    username = #{username},
    password = #{password},
    email = #{email},
    bio = #{bio}
  where id = #{id}
</update>

<delete id="deleteAuthor">
  delete from Author where id = #{id}
</delete>
```

如前所述，插入语句的配置规则更加丰富，在插入语句里面有一些额外的属性和子元素用来处理主键的生成，而且有多种生成方式。

首先，如果你的数据库支持自动生成主键的字段（比如 MySQL 和 SQL Server），那么你可以设置 useGeneratedKeys=”true”，然后再把 keyProperty 设置到目标属性上就OK了。例如，如果上面的 Author 表已经对 id 使用了自动生成的列类型，那么语句可以修改为:

```
<insert id="insertAuthor" useGeneratedKeys="true"
    keyProperty="id">
  insert into Author (username,password,email,bio)
  values (#{username},#{password},#{email},#{bio})
</insert>
```

对于不支持自动生成类型的数据库或可能不支持自动生成主键 JDBC 驱动来说，MyBatis 有另外一种方法来生成主键。

这里有一个简单（甚至很傻）的示例，它可以生成一个随机 ID（你最好不要这么做，但这里展示了 MyBatis 处理问题的灵活性及其所关心的广度）：

```
<insert id="insertAuthor">
  <selectKey keyProperty="id" resultType="int" order="BEFORE">
    select CAST(RANDOM()*1000000 as INTEGER) a from SYSIBM.SYSDUMMY
  </selectKey>
  insert into Author
    (id, username, password, email,bio, favourite_section)
  values
    (#{id}, #{username}, #{password}, #{email}, #{bio}, #{favourite
</insert>
```

在上面的示例中，selectKey 元素将会首先运行，Author 的 id 会被设置，然后插入语句会被调用。这给你了一个和数据库中来处理自动生成的主键类似的行为，避免了使 Java 代码变得复杂。

selectKey 元素描述如下：

```
<selectKey
  keyProperty="id"
  resultType="int"
  order="BEFORE"
  statementType="PREPARED">
```

selectKey 的属性

属性	描述
`keyProperty`	selectKey 语句结果应该被设置的目标属性。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
`keyColumn`	匹配属性的返回结果集中的列名称。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
`resultType`	结果的类型。MyBatis 通常可以推算出来，但是为了更加确定写上也不会有什么问题。MyBatis 允许任何简单类型用作主键的类型，包括字符串。如果希望作用于多个生成的列，则可以使用一个包含期望属性的 Object 或一个 Map。
`order`	这可以被设置为 BEFORE 或 AFTER。如果设置为 BEFORE，那么它会首先选择主键，设置 keyProperty 然后执行插入语句。如果设置为 AFTER，那么先执行插入语句，然后是 selectKey 元素 - 这和像 Oracle 的数据库相似，在插入语句内部可能有嵌入索引调用。
`statementType`	与前面相同，MyBatis 支持 STATEMENT，PREPARED 和 CALLABLE 语句的映射类型，分别代表 PreparedStatement 和 CallableStatement 类型。

sql

这个元素可以被用来定义可重用的 SQL 代码段，可以包含在其他语句中。它可以被静态地(在加载参数) 参数化. 不同的属性值通过包含的实例变化. 比如：

```
<sql id="userColumns"> ${alias}.id,${alias}.username,${alias}.passw
```

这个 SQL 片段可以被包含在其他语句中，例如：

```
<select id="selectUsers" resultType="map">
  select
    <include refid="userColumns"><property name="alias" value="t1"/
    <include refid="userColumns"><property name="alias" value="t2"/
  from some_table t1
    cross join some_table t2
</select>
```

属性值可以用于包含的refid属性或者包含的字句里面的属性值，例如：

```
<sql id="sometable">
  ${prefix}Table
</sql>

<sql id="someinclude">
  from
    <include refid="${include_target}"/>
</sql>

<select id="select" resultType="map">
  select
    field1, field2, field3
  <include refid="someinclude">
    <property name="prefix" value="Some"/>
    <property name="include_target" value="sometable"/>
  </include>
</select>
```

参数（Parameters）

前面的所有语句中你所见到的都是简单参数的例子，实际上参数是 MyBatis 非常强大的元素，对于简单的做法，大概 90% 的情况参数都很少，比如：

```
<select id="selectUsers" resultType="User">
  select id, username, password
  from users
  where id = #{id}
</select>
```

上面的这个示例说明了一个非常简单的命名参数映射。参数类型被设置为 `int`，这样这个参数就可以被设置成任何内容。原生的类型或简单数据类型（比如整型和字符串）因为没有相关属性，它会完全用参数值来替代。然而，如果传入一个复杂的对象，行为就会有一点不同了。比如：

```
<insert id="insertUser" parameterType="User">
  insert into users (id, username, password)
  values (#{id}, #{username}, #{password})
</insert>
```

如果 User 类型的参数对象传递到了语句中，id、username 和 password 属性将会被查找，然后将它们的值传入预处理语句的参数中。

这点对于向语句中传参是比较好的而且又简单，不过参数映射的功能远不止于此。

首先，像 MyBatis 的其他部分一样，参数也可以指定一个特殊的数据类型。

```
#{property,javaType=int,jdbcType=NUMERIC}
```

像 MyBatis 的剩余部分一样，javaType 通常可以从参数对象中来去确定，前提是只要对象不是一个 HashMap。那么 javaType 应该被确定来保证使用正确类型处理器。

NOTE 如果 null 被当作值来传递，对于所有可能为空的列，JDBC Type 是需要的。你可以自己通过阅读预处理语句的 setNull() 方法的 JavaDocs 文档来研究这种情况。

为了以后定制类型处理方式，你也可以指定一个特殊的类型处理器类（或别名），比如：

```
#{age,javaType=int,jdbcType=NUMERIC,typeHandler=MyTypeHandler}
```

尽管看起来配置变得越来越繁琐，但实际上是很少去设置它们。

对于数值类型，还有一个小数保留位数的设置，来确定小数点后保留的位数。

```
#{height,javaType=double,jdbcType=NUMERIC,numericScale=2}
```

最后，mode 属性允许你指定 IN，OUT 或 INOUT 参数。如果参数为 OUT 或 INOUT，参数对象属性的真实值将会被改变，就像你在获取输出参数时所期望的那样。如果 mode 为 OUT（或 INOUT），而且 jdbcType 为 CURSOR(也就是 Oracle 的 REFCURSOR)，你必须指定一个 resultMap 来映射结果集到参数类型。要注意这里的 javaType 属性是可选的，如果左边的空白是 jdbcType 的 CURSOR 类型，它会自动地被设置为结果集。

```
#{department, mode=OUT, jdbcType=CURSOR, javaType=ResultSet, result
```

MyBatis 也支持很多高级的数据类型，比如结构体，但是当注册 out 参数时你必须告诉它语句类型名称。比如（再次提示，在实际中要像这样不能换行）：

```
#{middleInitial, mode=OUT, jdbcType=STRUCT, jdbcTypeName=MY_TYPE, r
```

尽管所有这些强大的选项很多时候你只简单指定属性名，其他的事情 MyBatis 会自己去推断，最多你需要为可能为空的列名指定 `jdbcType` 。

```
#{firstName}
#{middleInitial,jdbcType=VARCHAR}
#{lastName}
```

字符串替换

默认情况下,使用#{}格式的语法会导致 MyBatis 创建预处理语句属性并安全地设置值（比如?）。这样做更安全，更迅速，通常也是首选做法，不过有时你只是想直接在 SQL 语句中插入一个不改变的字符串。比如，像 ORDER BY，你可以这样来使用：

```
ORDER BY ${columnName}
```

这里 MyBatis 不会修改或转义字符串。

NOTE 以这种方式接受从用户输出的内容并提供给语句中不变的字符串是不安全的，会导致潜在的 SQL 注入攻击，因此要么不允许用户输入这些字段，要么自行转义并检验。

Result Maps

resultMap 元素是 MyBatis 中最重要最强大的元素。它就是让你远离 90%的需要从结果 集中取出数据的 JDBC 代码的那个东西, 而且在一些情形下允许你做一些 JDBC 不支持的事 情。 事实上, 编写相似于对复杂语句联合映射这些等同的代码, 也许可以跨过上千行的代码。 ResultMap 的设计就是简单语句不需要明确的结果映射,而很多复杂语句确实需要描述它们 的关系。

你已经看到简单映射语句的示例了,但没有明确的 resultMap。比如:

```
<select id="selectUsers" resultType="map">
  select id, username, hashedPassword
  from some_table
  where id = #{id}
</select>
```

这样一个语句简单作用于所有列被自动映射到 HashMap 的键上,这由 resultType 属性 指定。这在很多情况下是有用的,但是 HashMap 不能很好描述一个领域模型。那样你的应 用程序将会使用 JavaBeans 或 POJOs(Plain Old Java Objects,普通 Java 对象)来作为领域 模型。MyBatis 对两者都支持。看看下面这个 JavaBean:

```
package com.someapp.model;
public class User {
  private int id;
  private String username;
  private String hashedPassword;

  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  public String getUsername() {
    return username;
  }
  public void setUsername(String username) {
    this.username = username;
  }
  public String getHashedPassword() {
    return hashedPassword;
  }
  public void setHashedPassword(String hashedPassword) {
    this.hashedPassword = hashedPassword;
  }
}
```

基于 JavaBean 的规范,上面这个类有 3 个属性:id,username 和 hashedPassword。这些 在 select 语句中会精确匹配到列名。

这样的一个 JavaBean 可以被映射到结果集,就像映射到 HashMap 一样简单。

```
<select id="selectUsers" resultType="com.someapp.model.User">
  select id, username, hashedPassword
  from some_table
  where id = #{id}
</select>
```

要记住类型别名是你的伙伴。使用它们你可以不用输入类的全路径。比如:

```
<!-- In mybatis-config.xml file -->
<typeAlias type="com.someapp.model.User" alias="User"/>

<!-- In SQL Mapping XML file -->
<select id="selectUsers" resultType="User">
  select id, username, hashedPassword
  from some_table
  where id = #{id}
</select>
```

这些情况下,MyBatis 会在幕后自动创建一个 ResultMap,基于属性名来映射列到 JavaBean 的属性上。如果列名没有精确匹配,你可以在列名上使用 select 字句的别名(一个 基本的 SQL 特性)来匹配标签。比如:

```
<select id="selectUsers" resultType="User">
  select
    user_id             as "id",
    user_name           as "userName",
    hashed_password     as "hashedPassword"
  from some_table
  where id = #{id}
</select>
```

ResultMap 最优秀的地方你已经了解了很多了,但是你还没有真正的看到一个。这些简 单的示例不需要比你看到的更多东西。 只是出于示例的原因, 让我们来看看最后一个示例中 外部的 resultMap 是什么样子的,这也是解决列名不匹配的另外一种方式。

```
<resultMap id="userResultMap" type="User">
  <id property="id" column="user_id" />
  <result property="username" column="user_name"/>
  <result property="password" column="hashed_password"/>
</resultMap>
```

引用它的语句使用 resultMap 属性就行了(注意我们去掉了 resultType 属性)。比如:

```
<select id="selectUsers" resultMap="userResultMap">
  select user_id, user_name, hashed_password
  from some_table
  where id = #{id}
</select>
```

如果世界总是这么简单就好了。

高级结果映射

MyBatis 创建的一个想法:数据库不用永远是你想要的或需要它们是什么样的。而我们 最喜欢的数据库最好是第三范式或 BCNF 模式,但它们有时不是。如果可能有一个单独的 数据库映射,所有应用程序都可以使用它,这是非常好的,但有时也不是。结果映射就是 MyBatis 提供处理这个问题的答案。

比如,我们如何映射下面这个语句?

```
<!-- Very Complex Statement -->
<select id="selectBlogDetails" resultMap="detailedBlogResultMap">
  select
       B.id as blog_id,
       B.title as blog_title,
       B.author_id as blog_author_id,
       A.id as author_id,
       A.username as author_username,
       A.password as author_password,
       A.email as author_email,
       A.bio as author_bio,
       A.favourite_section as author_favourite_section,
       P.id as post_id,
       P.blog_id as post_blog_id,
       P.author_id as post_author_id,
       P.created_on as post_created_on,
       P.section as post_section,
       P.subject as post_subject,
       P.draft as draft,
       P.body as post_body,
       C.id as comment_id,
       C.post_id as comment_post_id,
       C.name as comment_name,
       C.comment as comment_text,
       T.id as tag_id,
       T.name as tag_name
  from Blog B
       left outer join Author A on B.author_id = A.id
       left outer join Post P on B.id = P.blog_id
       left outer join Comment C on P.id = C.post_id
       left outer join Post_Tag PT on PT.post_id = P.id
       left outer join Tag T on PT.tag_id = T.id
  where B.id = #{id}
</select>
```

你可能想把它映射到一个智能的对象模型,包含一个作者写的博客,有很多的博文,每篇博文有零条或多条的评论和标签。 下面是一个完整的复杂结果映射例子 (假设作者, 博客, 博文, 评论和标签都是类型的别名) 我们来看看, 。 但是不用紧张, 我们会一步一步来说明。 当天最初它看起来令人生畏,但实际上非常简单。

```
<!-- Very Complex Result Map -->
<resultMap id="detailedBlogResultMap" type="Blog">
  <constructor>
    <idArg column="blog_id" javaType="int"/>
  </constructor>
  <result property="title" column="blog_title"/>
  <association property="author" javaType="Author">
    <id property="id" column="author_id"/>
    <result property="username" column="author_username"/>
    <result property="password" column="author_password"/>
    <result property="email" column="author_email"/>
    <result property="bio" column="author_bio"/>
    <result property="favouriteSection" column="author_favourite_se
  </association>
  <collection property="posts" ofType="Post">
    <id property="id" column="post_id"/>
    <result property="subject" column="post_subject"/>
    <association property="author" javaType="Author"/>
    <collection property="comments" ofType="Comment">
      <id property="id" column="comment_id"/>
    </collection>
    <collection property="tags" ofType="Tag" >
      <id property="id" column="tag_id"/>
    </collection>
    <discriminator javaType="int" column="draft">
      <case value="1" resultType="DraftPost"/>
    </discriminator>
  </collection>
</resultMap>
```

resultMap 元素有很多子元素和一个值得讨论的结构。 下面是 resultMap 元素的概念视图

resultMap

- `constructor` - 类在实例化时,用来注入结果到构造方法中
  - `idArg` - ID 参数;标记结果作为 ID 可以帮助提高整体效能
  - `arg` - 注入到构造方法的一个普通结果
- `id` – 一个 ID 结果;标记结果作为 ID 可以帮助提高整体效能
- `result` – 注入到字段或 JavaBean 属性的普通结果
- `association` – 一个复杂的类型关联;许多结果将包成这种类型
  - 嵌入结果映射 – 结果映射自身的关联,或者参考一个
- `collection` – 复杂类型的集
  - 嵌入结果映射 – 结果映射自身的集,或者参考一个
- `discriminator` – 使用结果值来决定使用哪个结果映射
  - `case` – 基于某些值的结果映射
    - 嵌入结果映射 – 这种情形结果也映射它本身,因此可以包含很多相 同

的元素,或者它可以参照一个外部的结果映射。

ResultMap Attributes

Attribute	Description
`id`	A unique identifier in this namespace that can be used to reference this result map.
`type`	A fully qualified Java class name, or a type alias (see the table above for the list of built-in type aliases).
`autoMapping`	If present, MyBatis will enable or disable the automapping for this ResultMap. This attribute overrides the global autoMappingBehavior. Default: unset.

最佳实践 通常逐步建立结果映射。单元测试的真正帮助在这里。如果你尝试创建一次创建一个向上面示例那样的巨大的结果映射, 那么可能会有错误而且很难去控制它 来工作。开始简单一些,一步一步的发展。而且要进行单元测试!使用该框架的缺点是 它们有时是黑盒(是否可见源代码) 。你确定你实现想要的行为的最好选择是编写单元 测试。它也可以你帮助得到提交时的错误。

下面一部分将详细说明每个元素。

id & result

```
<id property="id" column="post_id"/>
<result property="subject" column="post_subject"/>
```

这些是结果映射最基本内容。id 和 result 都映射一个单独列的值到简单数据类型(字符 串,整型,双精度浮点数,日期等)的单独属性或字段。

这两者之间的唯一不同是 id 表示的结果将是当比较对象实例时用到的标识属性。这帮 助来改进整体表现,特别是缓存和嵌入结果映射(也就是联合映射) 。

每个都有一些属性:

Id and Result Attributes

属性	描述
`property`	映射到列结果的字段或属性。如果匹配的是存在的,和给定名称相同 的 JavaBeans 的属性,那么就会使用。否则 MyBatis 将会寻找给定名称 property 的字段。这两种情形你可以使用通常点式的复杂属性导航。比如,你 可以这样映射一些东西: “username” ,或者映射到一些复杂的东西: “address.street.number” 。
`column`	从数据库中得到的列名,或者是列名的重命名标签。这也是通常和会 传递给 resultSet.getString(columnName)方法参数中相同的字符串。
`javaType`	一个 Java 类的完全限定名,或一个类型别名(参考上面内建类型别名 的列表) 。如果你映射到一个 JavaBean,MyBatis 通常可以断定类型。 然而,如果你映射到的是 HashMap,那么你应该明确地指定 javaType 来保证所需的行为。
`jdbcType`	在这个表格之后的所支持的 JDBC 类型列表中的类型。JDBC 类型是仅 仅需要对插入,更新和删除操作可能为空的列进行处理。这是 JDBC jdbcType 的需要,而不是 MyBatis 的。如果你直接使用 JDBC 编程,你需要指定 这个类型-但仅仅对可能为空的值。
`typeHandler`	我们在前面讨论过默认的类型处理器。使用这个属性,你可以覆盖默 认的类型处理器。这个属性值是类的完全限定名或者是一个类型处理 器的实现,或者是类型别名。

支持的 JDBC 类型

为了未来的参考,MyBatis 通过包含的 jdbcType 枚举型,支持下面的 JDBC 类型。

`BIT`	`FLOAT`	`CHAR`	`TIMESTAMP`	`OTHER`
`TINYINT`	`REAL`	`VARCHAR`	`BINARY`	`BLOG`
`SMALLINT`	`DOUBLE`	`LONGVARCHAR`	`VARBINARY`	`CLOB`
`INTEGER`	`NUMERIC`	`DATE`	`LONGVARBINARY`	`BOOLEAN`
`BIGINT`	`DECIMAL`	`TIME`	`NULL`	`CURSOR`

构造方法

```
<constructor>
   <idArg column="id" javaType="int"/>
   <arg column="username" javaType="String"/>
</constructor>
```

对于大多数数据传输对象(Data Transfer Object,DTO)类型,属性可以起作用,而且像你绝大多数的领域模型, 指令也许是你想使用一成不变的类的地方。 通常包含引用或查询数 据的表很少或基本不变的话对一成不变的类来说是合适的。 构造方法注入允许你在初始化时 为类设置属性的值,而不用暴露出公有方法。MyBatis 也支持私有属性和私有 JavaBeans 属 性来达到这个目的,但是一些人更青睐构造方法注入。构造方法元素支持这个。

看看下面这个构造方法:

```
public class User {
   //...
   public User(int id, String username) {
     //...
  }
//...
}
```

为了向这个构造方法中注入结果,MyBatis 需要通过它的参数的类型来标识构造方法。 Java 没有自查(反射)参数名的方法。所以当创建一个构造方法元素时,保证参数是按顺序 排列的,而且数据类型也是确定的。

```
<constructor>
   <idArg column="id" javaType="int"/>
   <arg column="username" javaType="String"/>
</constructor>
```

剩余的属性和规则和固定的 id 和 result 元素是相同的。

属性	描述
`column`	来自数据库的类名,或重命名的列标签。这和通常传递给 resultSet.getString(columnName)方法的字符串是相同的。
`javaType`	一个 Java 类的完全限定名,或一个类型别名(参考上面内建类型别名的列表)。 如果你映射到一个 JavaBean,MyBatis 通常可以断定类型。然而,如 果你映射到的是 HashMap,那么你应该明确地指定 javaType 来保证所需的 行为。
`jdbcType`	在这个表格之前的所支持的 JDBC 类型列表中的类型。JDBC 类型是仅仅 需要对插入, 更新和删除操作可能为空的列进行处理。这是 JDBC 的需要, jdbcType 而不是 MyBatis 的。如果你直接使用 JDBC 编程,你需要指定这个类型-但 仅仅对可能为空的值。
`typeHandler`	我们在前面讨论过默认的类型处理器。使用这个属性,你可以覆盖默认的 类型处理器。 这个属性值是类的完全限定名或者是一个类型处理器的实现, 或者是类型别名。
`select`	The ID of another mapped statement that will load the complex type required by this property mapping. The values retrieved from columns specified in the column attribute will be passed to the target select statement as parameters. See the Association element for more.
`resultMap`	This is the ID of a ResultMap that can map the nested results of this argument into an appropriate object graph. This is an alternative to using a call to another select statement. It allows you to join multiple tables together into a single `ResultSet` . Such a `ResultSet` will contain duplicated, repeating groups of data that needs to be decomposed and mapped properly to a nested object graph. To facilitate this, MyBatis lets you "chain" result maps together, to deal with the nested results. See the Association element below for more.

关联

```
<association property="author" column="blog_author_id" javaType="Au
  <id property="id" column="author_id"/>
  <result property="username" column="author_username"/>
</association>
```

关联元素处理“有一个”类型的关系。比如,在我们的示例中,一个博客有一个用户。关联映射就工作于这种结果之上。你指定了目标属性,来获取值的列,属性的 java 类型(很 多情况下 MyBatis 可以自己算出来) ,如果需要的话还有 jdbc 类型,如果你想覆

盖或获取的 结果值还需要类型控制器。

关联中不同的是你需要告诉 MyBatis 如何加载关联。MyBatis 在这方面会有两种不同的 方式:

- 嵌套查询:通过执行另外一个 SQL 映射语句来返回预期的复杂类型。
- 嵌套结果:使用嵌套结果映射来处理重复的联合结果的子集。首先,然让我们来查看这个元素的属性。所有的你都会看到,它和普通的只由 select 和

resultMap 属性的结果映射不同。

属性	描述
`property`	映射到列结果的字段或属性。如果匹配的是存在的,和给定名称相同的 property JavaBeans 的属性, 那么就会使用。 否则 MyBatis 将会寻找给定名称的字段。 这两种情形你可以使用通常点式的复杂属性导航。比如,你可以这样映射 一 些 东 西 :“ username ”, 或 者 映 射 到 一 些 复 杂 的 东 西 : “address.street.number” 。
`javaType`	一个 Java 类的完全限定名,或一个类型别名(参考上面内建类型别名的列 表) 。如果你映射到一个 JavaBean,MyBatis 通常可以断定类型。然而,如 javaType 果你映射到的是 HashMap,那么你应该明确地指定 javaType 来保证所需的 行为。
`jdbcType`	在这个表格之前的所支持的 JDBC 类型列表中的类型。JDBC 类型是仅仅 需要对插入, 更新和删除操作可能为空的列进行处理。这是 JDBC 的需要, jdbcType 而不是 MyBatis 的。如果你直接使用 JDBC 编程,你需要指定这个类型-但 仅仅对可能为空的值。
`typeHandler`	我们在前面讨论过默认的类型处理器。使用这个属性,你可以覆盖默认的 typeHandler 类型处理器。 这个属性值是类的完全限定名或者是一个类型处理器的实现, 或者是类型别名。

关联的嵌套查询

属性	描述
`column`	来自数据库的类名,或重命名的列标签。这和通常传递给 resultSet.getString(columnName)方法的字符串是相同的。 column 注 意 : 要 处 理 复 合 主 键 , 你 可 以 指 定 多 个 列 名 通 过 column= ” {prop1=col1,prop2=col2} ” 这种语法来传递给嵌套查询语 句。这会引起 prop1 和 prop2 以参数对象形式来设置给目标嵌套查询语句。
`select`	另外一个映射语句的 ID,可以加载这个属性映射需要的复杂类型。获取的 在列属性中指定的列的值将被传递给目标 select 语句作为参数。表格后面 有一个详细的示例。 select 注 意 : 要 处 理 复 合 主 键 , 你 可 以 指 定 多 个 列 名 通 过 column= ” {prop1=col1,prop2=col2} ” 这种语法来传递给嵌套查询语 句。这会引起 prop1 和 prop2 以参数对象形式来设置给目标嵌套查询语句。
`fetchType`	Optional. Valid values are `lazy` and `eager`. If present, it supersedes the global configuration parameter `lazyLoadingEnabled` for this mapping.

示例:

```
<resultMap id="blogResult" type="Blog">
  <association property="author" column="author_id" javaType="Autho
</resultMap>

<select id="selectBlog" resultMap="blogResult">
  SELECT * FROM BLOG WHERE ID = #{id}
</select>

<select id="selectAuthor" resultType="Author">
  SELECT * FROM AUTHOR WHERE ID = #{id}
</select>
```

我们有两个查询语句:一个来加载博客,另外一个来加载作者,而且博客的结果映射描述了“selectAuthor”语句应该被用来加载它的 author 属性。

其他所有的属性将会被自动加载,假设它们的列和属性名相匹配。

这种方式很简单, 但是对于大型数据集合和列表将不会表现很好。 问题就是我们熟知的 “N+1 查询问题”。概括地讲,N+1 查询问题可以是这样引起的:

- 你执行了一个单独的 SQL 语句来获取结果列表(就是“+1”)。
- 对返回的每条记录,你执行了一个查询语句来为每个加载细节(就是“N”)。

这个问题会导致成百上千的 SQL 语句被执行。这通常不是期望的。

MyBatis 能延迟加载这样的查询就是一个好处,因此你可以分散这些语句同时运行的消 耗。然而,如果你加载一个列表,之后迅速迭代来访问嵌套的数据,你会调用所有的延迟加 载,这样的行为可能是很糟糕的。

所以还有另外一种方法。

关联的嵌套结果

属性	描述
`resultMap`	这是结果映射的 ID,可以映射关联的嵌套结果到一个合适的对象图中。这 是一种替代方法来调用另外一个查询语句。这允许你联合多个表来合成到 resultMap 一个单独的结果集。这样的结果集可能包含重复,数据的重复组需要被分 解,合理映射到一个嵌套的对象图。为了使它变得容易,MyBatis 让你“链 接”结果映射,来处理嵌套结果。一个例子会很容易来仿照,这个表格后 面也有一个示例。
`columnPrefix`	When joining multiple tables, you would have to use column alias to avoid duplicated column names in the ResultSet. Specifying columnPrefix allows you to map such columns to an external resultMap. Please see the example explained later in this section.
`notNullColumn`	By default a child object is created only if at least one of the columns mapped to the child's properties is non null. With this attribute you can change this behaviour by specifiying which columns must have a value so MyBatis will create a child object only if any of those columns is not null. Multiple column names can be specified using a comma as a separator. Default value: unset.
`autoMapping`	If present, MyBatis will enable or disable auto-mapping when mapping the result to this property. This attribute overrides the global autoMappingBehavior. Note that it has no effect on an external resultMap, so it is pointless to use it with `select` or `resultMap` attribute. Default value: unset.

在上面你已经看到了一个非常复杂的嵌套关联的示例。 下面这个是一个非常简单的示例 来说明它如何工作。代替了执行一个分离的语句,我们联合博客表和作者表在一起,就像:

```
<select id="selectBlog" resultMap="blogResult">
  select
    B.id            as blog_id,
    B.title         as blog_title,
    B.author_id     as blog_author_id,
    A.id            as author_id,
    A.username      as author_username,
    A.password      as author_password,
    A.email         as author_email,
    A.bio           as author_bio
  from Blog B left outer join Author A on B.author_id = A.id
  where B.id = #{id}
</select>
```

注意这个联合查询, 以及采取保护来确保所有结果被唯一而且清晰的名字来重命名。 这使得映射非常简单。现在我们可以映射这个结果:

```
<resultMap id="blogResult" type="Blog">
  <id property="id" column="blog_id" />
  <result property="title" column="blog_title"/>
  <association property="author" column="blog_author_id" javaType='
</resultMap>

<resultMap id="authorResult" type="Author">
  <id property="id" column="author_id"/>
  <result property="username" column="author_username"/>
  <result property="password" column="author_password"/>
  <result property="email" column="author_email"/>
  <result property="bio" column="author_bio"/>
</resultMap>
```

在上面的示例中你可以看到博客的作者关联代表着“authorResult”结果映射来加载作者实例。

非常重要: 在嵌套据诶过映射中 id 元素扮演了非常重要的角色。应应该通常指定一个 或多个属性,它们可以用来唯一标识结果。实际上就是如果你离开她了,但是有一个严重的 性能问题时 MyBatis 仍然可以工作。选择的属性越少越好,它们可以唯一地标识结果。主键 就是一个显而易见的选择(尽管是联合主键)。

现在,上面的示例用了外部的结果映射元素来映射关联。这使得 Author 结果映射可以 重用。然而,如果你不需要重用它的话,或者你仅仅引用你所有的结果映射合到一个单独描 述的结果映射中。你可以嵌套结果映射。这里给出使用这种方式的相同示例:

```
<resultMap id="blogResult" type="Blog">
  <id property="id" column="blog_id" />
  <result property="title" column="blog_title"/>
  <association property="author" javaType="Author">
    <id property="id" column="author_id"/>
    <result property="username" column="author_username"/>
    <result property="password" column="author_password"/>
    <result property="email" column="author_email"/>
    <result property="bio" column="author_bio"/>
  </association>
</resultMap>
```

What if the blog has a co-author? The select statement would look like:

```
<select id="selectBlog" resultMap="blogResult">
  select
    B.id            as blog_id,
    B.title         as blog_title,
    A.id            as author_id,
    A.username      as author_username,
    A.password      as author_password,
    A.email         as author_email,
    A.bio           as author_bio,
    CA.id           as co_author_id,
    CA.username     as co_author_username,
    CA.password     as co_author_password,
    CA.email        as co_author_email,
    CA.bio          as co_author_bio
  from Blog B
  left outer join Author A on B.author_id = A.id
  left outer join Author CA on B.co_author_id = CA.id
  where B.id = #{id}
</select>
```

Recall that the resultMap for Author is defined as follows.

```
<resultMap id="authorResult" type="Author">
  <id property="id" column="author_id"/>
  <result property="username" column="author_username"/>
  <result property="password" column="author_password"/>
  <result property="email" column="author_email"/>
  <result property="bio" column="author_bio"/>
</resultMap>
```

Because the column names in the results differ from the columns defined in the resultMap, you need to specify `columnPrefix` to reuse the resultMap for mapping co-author results.

```
<resultMap id="blogResult" type="Blog">
  <id property="id" column="blog_id" />
  <result property="title" column="blog_title"/>
  <association property="author"
    resultMap="authorResult" />
  <association property="coAuthor"
    resultMap="authorResult"
    columnPrefix="co_" />
</resultMap>
```

上面你已经看到了如何处理“有一个”类型关联。但是“有很多个”是怎样的?下面这 个部分就是来讨论这个主题的。

集合

```
<collection property="posts" ofType="domain.blog.Post">
  <id property="id" column="post_id"/>
  <result property="subject" column="post_subject"/>
  <result property="body" column="post_body"/>
</collection>
```

集合元素的作用几乎和关联是相同的。实际上,它们也很相似,文档的异同是多余的。 所以我们更多关注于它们的不同。

我们来继续上面的示例,一个博客只有一个作者。但是博客有很多文章。在博客类中, 这可以由下面这样的写法来表示:

```
private List<Post> posts;
```

要映射嵌套结果集合到 List 中,我们使用集合元素。就像关联元素一样,我们可以从连接中使用嵌套查询,或者嵌套结果。

集合的嵌套查询

首先,让我们看看使用嵌套查询来为博客加载文章。

```
<resultMap id="blogResult" type="Blog">
  <collection property="posts" javaType="ArrayList" column="id" of
</resultMap>

<select id="selectBlog" resultMap="blogResult">
  SELECT * FROM BLOG WHERE ID = #{id}
</select>

<select id="selectPostsForBlog" resultType="Blog">
  SELECT * FROM POST WHERE BLOG_ID = #{id}
</select>
```

这里你应该注意很多东西,但大部分代码和上面的关联元素是非常相似的。首先,你应 该注意我们使用的是集合元素。然后要注意那个新的“ofType”属性。这个属性用来区分 JavaBean(或字段)属性类型和集合包含的类型来说是很重要的。所以你可以读出下面这个 映射:

```
<collection property="posts" javaType="ArrayList" column="id" ofTy
```

读作: “在 Post 类型的 ArrayList 中的 posts 的集合。”

javaType 属性是不需要的,因为 MyBatis 在很多情况下会为你算出来。所以你可以缩短 写法:

```
<collection property="posts" column="id" ofType="Post" select="sele
```

集合的嵌套结果

至此,你可以猜测集合的嵌套结果是如何来工作的,因为它和关联完全相同,除了它应用了一个“ofType”属性

First, let's look at the SQL:

```
<select id="selectBlog" resultMap="blogResult">
  select
  B.id as blog_id,
  B.title as blog_title,
  B.author_id as blog_author_id,
  P.id as post_id,
  P.subject as post_subject,
  P.body as post_body,
  from Blog B
  left outer join Post P on B.id = P.blog_id
  where B.id = #{id}
</select>
```

我们又一次联合了博客表和文章表,而且关注于保证特性,结果列标签的简单映射。现 在用文章映射集合映射博客,可以简单写为:

```
<resultMap id="blogResult" type="Blog">
  <id property="id" column="blog_id" />
  <result property="title" column="blog_title"/>
  <collection property="posts" ofType="Post">
    <id property="id" column="post_id"/>
    <result property="subject" column="post_subject"/>
    <result property="body" column="post_body"/>
  </collection>
</resultMap>
```

同样,要记得 id 元素的重要性,如果你不记得了,请阅读上面的关联部分。

同样, 如果你引用更长的形式允许你的结果映射的更多重用, 你可以使用下面这个替代 的映射:

```
<resultMap id="blogResult" type="Blog">
  <id property="id" column="blog_id" />
  <result property="title" column="blog_title"/>
  <collection property="posts" ofType="Post" resultMap="blogPostRes
</resultMap>

<resultMap id="blogPostResult" type="Post">
  <id property="id" column="id"/>
  <result property="subject" column="subject"/>
  <result property="body" column="body"/>
</resultMap>
```

注意 这个对你所映射的内容没有深度,广度或关联和集合相联合的限制。当映射它们 时你应该在大脑中保留它们的表现。 你的应用在找到最佳方法前要一直进行的单元测试和性 能测试。好在 myBatis 让你后来可以改变想法,而不对你的代码造成很小(或任何)影响。

高级关联和集合映射是一个深度的主题。文档只能给你介绍到这了。加上一点联系,你 会很快清楚它们的用法。

鉴别器

```
<discriminator javaType="int" column="draft">
  <case value="1" resultType="DraftPost"/>
</discriminator>
```

有时一个单独的数据库查询也许返回很多不同 (但是希望有些关联) 数据类型的结果集。 鉴别器元素就是被设计来处理这个情况的, 还有包括类的继承层次结构。 鉴别器非常容易理 解,因为它的表现很像 Java 语言中的 switch 语句。

定义鉴别器指定了 column 和 javaType 属性。 列是 MyBatis 查找比较值的地方。 JavaType 是需要被用来保证等价测试的合适类型(尽管字符串在很多情形下都会有用)。比如:

```
<resultMap id="vehicleResult" type="Vehicle">
  <id property="id" column="id" />
  <result property="vin" column="vin"/>
  <result property="year" column="year"/>
  <result property="make" column="make"/>
  <result property="model" column="model"/>
  <result property="color" column="color"/>
  <discriminator javaType="int" column="vehicle_type">
    <case value="1" resultMap="carResult"/>
    <case value="2" resultMap="truckResult"/>
    <case value="3" resultMap="vanResult"/>
    <case value="4" resultMap="suvResult"/>
  </discriminator>
</resultMap>
```

在这个示例中, MyBatis 会从结果集中得到每条记录, 然后比较它的 vehicle 类型的值。 如果它匹配任何一个鉴别器的实例,那么就使用这个实例指定的结果映射。换句话说,这样 做完全是剩余的结果映射被忽略(除非它被扩展,这在第二个示例中讨论) 。如果没有任何 一个实例相匹配,那么 MyBatis 仅仅使用鉴别器块外定义的结果映射。所以,如果 carResult 按如下声明:

```
<resultMap id="carResult" type="Car">
  <result property="doorCount" column="door_count" />
</resultMap>
```

那么只有 doorCount 属性会被加载。这步完成后完整地允许鉴别器实例的独立组,尽管 和父结果映射可能没有什么关系。这种情况下,我们当然知道 cars 和 vehicles 之间有关系, 如 Car 是一个 Vehicle 实例。因此,我们想要剩余的属性也被加载。我们设置的结果映射的 简单改变如下。

```
<resultMap id="carResult" type="Car" extends="vehicleResult">
  <result property="doorCount" column="door_count" />
</resultMap>
```

现在 vehicleResult 和 carResult 的属性都会被加载了。

尽管曾经有些人会发现这个外部映射定义会多少有一些令人厌烦之处。 因此还有另外一 种语法来做简洁的映射风格。比如:

```
<resultMap id="vehicleResult" type="Vehicle">
  <id property="id" column="id" />
  <result property="vin" column="vin"/>
  <result property="year" column="year"/>
  <result property="make" column="make"/>
  <result property="model" column="model"/>
  <result property="color" column="color"/>
  <discriminator javaType="int" column="vehicle_type">
    <case value="1" resultType="carResult">
      <result property="doorCount" column="door_count" />
    </case>
    <case value="2" resultType="truckResult">
      <result property="boxSize" column="box_size" />
      <result property="extendedCab" column="extended_cab" />
    </case>
    <case value="3" resultType="vanResult">
      <result property="powerSlidingDoor" column="power_sliding_doo
    </case>
    <case value="4" resultType="suvResult">
      <result property="allWheelDrive" column="all_wheel_drive" />
    </case>
  </discriminator>
</resultMap>
```

要记得 这些都是结果映射, 如果你不指定任何结果, 那么 MyBatis 将会为你自动匹配列 和属性。所以这些例子中的大部分是很冗长的,而其实是不需要的。也就是说,很多数据库 是很复杂的,我们不太可能对所有示例都能依靠它。

自动映射

正如你在前面一节看到的，在简单的场景下，MyBatis可以替你自动映射查询结果。 如果遇到复杂的场景，你需要构建一个result map。 但是在本节你将看到，你也可以混合使用这两种策略。 让我们到深一点的层面上看看自动映射是怎样工作的。

当自动映射查询结果时，MyBatis会获取sql返回的列名并在java类中查找相同名字的属性（忽略大小写）。 这意味着如果Mybatis发现了*ID*列和*id*属性，Mybatis会将*ID*的值赋给*id*。

通常数据库列使用大写单词命名，单词间用下划线分隔；而java属性一般遵循驼峰命名法。为了在这两种命名方式之间启用自动映射，需要将 `mapUnderscoreToCamelCase` 设置为true。

自动映射甚至在特定的result map下也能工作。在这种情况下，对于每一个result map,所有的ResultSet提供的列，如果没有被手工映射，则将被自动映射。自动映射处理完毕后手工映射才会被处理。在接下来的例子中，*id* 和 *userName*列将被自动映射，*hashed_password* 列将根据配置映射。

```
<select id="selectUsers" resultMap="userResultMap">
  select
    user_id             as "id",
    user_name           as "userName",
    hashed_password
  from some_table
  where id = #{id}
</select>
```

```
<resultMap id="userResultMap" type="User">
  <result property="password" column="hashed_password"/>
</resultMap>
```

There are three auto-mapping levels:

- `NONE` - disables auto-mapping. Only manually mapped properties will be set.
- `PARTIAL` - will auto-map results except those that have nested result mappings defined inside (joins).
- `FULL` - auto-maps everything.

The default value is `PARTIAL`, and it is so for a reason. When `FULL` is used auto-mapping will be performed when processing join results and joins retrieve data of several different entities in the same row hence this may result in undesired mappings. To understand the risk have a look at the following sample:

```
<select id="selectBlog" resultMap="blogResult">
  select
    B.id,
    B.title,
    A.username,
  from Blog B left outer join Author A on B.author_id = A.id
  where B.id = #{id}
</select>
```

```
<resultMap id="blogResult" type="Blog">
  <association property="author" resultMap="authorResult"/>
</resultMap>

<resultMap id="authorResult" type="Author">
  <result property="username" column="author_username"/>
</resultMap>
```

With this result map both *Blog* and *Author* will be auto-mapped. But note that *Author* has an *id* property and there is a column named *id* in the ResultSet so Author's id will be filled with Blog's id, and that is not what you were expecting. So use the `FULL` option with caution.

Regardless of the auto-mapping level configured you can enable or disable the automapping for an specific ResultMap by adding the attribute `autoMapping` to it:

```
<resultMap id="userResultMap" type="User" autoMapping="false">
  <result property="password" column="hashed_password"/>
</resultMap>
```

缓存

MyBatis 包含一个非常强大的查询缓存特性,它可以非常方便地配置和定制。MyBatis 3 中的缓存实现的很多改进都已经实现了,使得它更加强大而且易于配置。

默认情况下是没有开启缓存的,除了局部的 session 缓存,可以增强变现而且处理循环 依赖也是必须的。要开启二级缓存,你需要在你的 SQL 映射文件中添加一行:

```
<cache/>
```

字面上看就是这样。这个简单语句的效果如下:

- 映射语句文件中的所有 select 语句将会被缓存。
- 映射语句文件中的所有 insert,update 和 delete 语句会刷新缓存。
- 缓存会使用 Least Recently Used(LRU,最近最少使用的)算法来收回。
- 根据时间表(比如 no Flush Interval,没有刷新间隔), 缓存不会以任何时间顺序来刷新。
- 缓存会存储列表集合或对象(无论查询方法返回什么)的 1024 个引用。
- 缓存会被视为是 read/write(可读/可写)的缓存,意味着对象检索不是共享的,而且可以安全地被调用者修改,而不干扰其他调用者或线程所做的潜在修改。

所有的这些属性都可以通过缓存元素的属性来修改。比如:

```
<cache
  eviction="FIFO"
  flushInterval="60000"
  size="512"
  readOnly="true"/>
```

这个更高级的配置创建了一个 FIFO 缓存,并每隔 60 秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的,因此在不同线程中的调用者之间修改它们会 导致冲突。

可用的收回策略有:

- `LRU` – 最近最少使用的:移除最长时间不被使用的对象。
- `FIFO` – 先进先出:按对象进入缓存的顺序来移除它们。
- `SOFT` – 软引用:移除基于垃圾回收器状态和软引用规则的对象。
- `WEAK` – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象。

默认的是 LRU。

flushInterval(刷新间隔)可以被设置为任意的正整数,而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置,也就是没有刷新间隔,缓存仅仅调用语句时刷新。

size(引用数目)可以被设置为任意正整数,要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是 1024。

readOnly(只读)属性可以被设置为 true 或 false。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存 会返回缓存对象的拷贝(通过序列化) 。这会慢一些,但是安全,因此默认是 false。

使用自定义缓存

除了这些自定义缓存的方式, 你也可以通过实现你自己的缓存或为其他第三方缓存方案 创建适配器来完全覆盖缓存行为。

```
<cache type="com.domain.something.MyCustomCache"/>
```

这个示 例展 示了 如何 使用 一个 自定义 的缓 存实 现。type 属 性指 定的 类必 须实现 org.mybatis.cache.Cache 接口。这个接口是 MyBatis 框架中很多复杂的接口之一,但是简单 给定它做什么就行。

```
public interface Cache {
  String getId();
  int getSize();
  void putObject(Object key, Object value);
  Object getObject(Object key);
  boolean hasKey(Object key);
  Object removeObject(Object key);
  void clear();
}
```

要配置你的缓存, 简单和公有的 JavaBeans 属性来配置你的缓存实现, 而且是通过 cache 元素来传递属性, 比如, 下面代码会在你的缓存实现中调用一个称为 “setCacheFile(String file)” 的方法:

```
<cache type="com.domain.something.MyCustomCache">
  <property name="cacheFile" value="/tmp/my-custom-cache.tmp"/>
</cache>
```

你可以使用所有简单类型作为 JavaBeans 的属性,MyBatis 会进行转换。

记得缓存配置和缓存实例是绑定在 SQL 映射文件的命名空间是很重要的。因此,所有 在相同命名空间的语句正如绑定的缓存一样。 语句可以修改和缓存交互的方式,或在语句的 语句的基础上使用两种简单的属性来完全排除它们。默认情况下,语句可以这样来配置:

```
<select ... flushCache="false" useCache="true"/>
<insert ... flushCache="true"/>
<update ... flushCache="true"/>
<delete ... flushCache="true"/>
```

因为那些是默认的,你明显不能明确地以这种方式来配置一条语句。相反,如果你想改 变默认的行为,只能设置 flushCache 和 useCache 属性。比如,在一些情况下你也许想排除 从缓存中查询特定语句结果,或者你也许想要一个查询语句来刷新缓存。相似地,你也许有 一些更新语句依靠执行而不需要刷新缓存。

参照缓存

回想一下上一节内容, 这个特殊命名空间的唯一缓存会被使用或者刷新相同命名空间内 的语句。也许将来的某个时候,你会想在命名空间中共享相同的缓存配置和实例。在这样的 情况下你可以使用 cache-ref 元素来引用另外一个缓存。

```
<cache-ref namespace="com.someone.application.data.SomeMapper"/>
```

动态 SQL

MyBatis 的强大特性之一便是它的动态 SQL。如果你有使用 JDBC 或其他类似框架的经验，你就能体会到根据不同条件拼接 SQL 语句有多么痛苦。拼接的时候要确保不能忘了必要的空格，还要注意省掉列名列表最后的逗号。利用动态 SQL 这一特性可以彻底摆脱这种痛苦。

通常使用动态 SQL 不可能是独立的一部分,MyBatis 当然使用一种强大的动态 SQL 语言来改进这种情形,这种语言可以被用在任意的 SQL 映射语句中。

动态 SQL 元素和使用 JSTL 或其他类似基于 XML 的文本处理器相似。在 MyBatis 之前的版本中,有很多的元素需要来了解。MyBatis 3 大大提升了它们,现在用不到原先一半的元素就可以了。MyBatis 采用功能强大的基于 OGNL 的表达式来消除其他元素。

- if
- choose (when, otherwise)
- trim (where, set)
- foreach

if

动态 SQL 通常要做的事情是有条件地包含 where 子句的一部分。比如:

```
<select id="findActiveBlogWithTitleLike"
     resultType="Blog">
  SELECT * FROM BLOG
  WHERE state = ‘ACTIVE’
  <if test="title != null">
    AND title like #{title}
  </if>
</select>
```

这条语句提供了一个可选的文本查找类型的功能。如果没有传入“title”，那么所有处于“ACTIVE”状态的BLOG都会返回；反之若传入了“title”，那么就会把模糊查找“title”内容的BLOG结果返回（就这个例子而言，细心的读者会发现其中的参数值是可以包含一些掩码或通配符的）。

如果想可选地通过“title”和“author”两个条件搜索该怎么办呢？首先，改变语句的名称让它更具实际意义；然后只要加入另一个条件即可。

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG WHERE state = 'ACTIVE'
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

choose, when, otherwise

有些时候，我们不想用到所有的条件语句，而只想从中择其一二。针对这种情况，MyBatis 提供了 choose 元素，它有点像 Java 中的 switch 语句。

还是上面的例子，但是这次变为提供了“title”就按“title”查找，提供了“author”就按“author”查找，若两者都没有提供，就返回所有符合条件的BLOG（实际情况可能是由管理员按一定策略选出BLOG列表，而不是返回大量无意义的随机结果）。

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG WHERE state = 'ACTIVE'
  <choose>
    <when test="title != null">
      AND title like #{title}
    </when>
    <when test="author != null and author.name != null">
      AND author_name like #{author.name}
    </when>
    <otherwise>
      AND featured = 1
    </otherwise>
  </choose>
</select>
```

trim, where, set

前面几个例子已经合宜地解决了一个臭名昭著的动态 SQL 问题。现在考虑回到“if”示例，这次我们将“ACTIVE = 1”也设置成动态的条件，看看会发生什么。

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  WHERE
  <if test="state != null">
    state = #{state}
  </if>
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

如果这些条件没有一个能匹配上将会怎样？最终这条 SQL 会变成这样：

```
SELECT * FROM BLOG
WHERE
```

这会导致查询失败。如果仅仅第二个条件匹配又会怎样？这条 SQL 最终会是这样:

```
SELECT * FROM BLOG
WHERE
AND title like ‘someTitle’
```

这个查询也会失败。这个问题不能简单的用条件句式来解决，如果你也曾经被迫这样写过，那么你很可能从此以后都不想再这样去写了。

MyBatis 有一个简单的处理，这在90%的情况下都会有用。而在不能使用的地方，你可以自定义处理方式来令其正常工作。一处简单的修改就能得到想要的效果：

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  <where>
    <if test="state != null">
         state = #{state}
    </if>
    <if test="title != null">
        AND title like #{title}
    </if>
    <if test="author != null and author.name != null">
        AND author_name like #{author.name}
    </if>
  </where>
</select>
```

where 元素知道只有在一个以上的if条件有值的情况下才去插入“WHERE”子句。而且，若最后的内容是“AND”或“OR”开头的，where 元素也知道如何将他们去除。

如果 where 元素没有按正常套路出牌，我们还是可以通过自定义 trim 元素来定制我们想要的功能。比如，和 where 元素等价的自定义 trim 元素为：

```
<trim prefix="WHERE" prefixOverrides="AND |OR ">
  ...
</trim>
```

prefixOverrides 属性会忽略通过管道分隔的文本序列（注意此例中的空格也是必要的）。它带来的结果就是所有在 prefixOverrides 属性中指定的内容将被移除，并且插入 prefix 属性中指定的内容。

类似的用于动态更新语句的解决方案叫做 set。set 元素可以被用于动态包含需要更新的列，而舍去其他的。比如：

```
<update id="updateAuthorIfNecessary">
  update Author
    <set>
      <if test="username != null">username=#{username},</if>
      <if test="password != null">password=#{password},</if>
      <if test="email != null">email=#{email},</if>
      <if test="bio != null">bio=#{bio}</if>
    </set>
  where id=#{id}
</update>
```

这里，set 元素会动态前置 SET 关键字，同时也会消除无关的逗号，因为用了条件语句之后很可能就会在生成的赋值语句的后面留下这些逗号。

若你对等价的自定义 trim 元素的样子感兴趣，那这就应该是它的真面目：

```
<trim prefix="SET" suffixOverrides=",">
  ...
</trim>
```

注意这里我们忽略的是后缀中的值，而又一次附加了前缀中的值。

foreach

动态 SQL 的另外一个常用的必要操作是需要对一个集合进行遍历，通常是在构建 IN 条件语句的时候。比如：

```
<select id="selectPostIn" resultType="domain.blog.Post">
  SELECT *
  FROM POST P
  WHERE ID in
  <foreach item="item" index="index" collection="list"
      open="(" separator="," close=")">
        #{item}
  </foreach>
</select>
```

foreach 元素的功能是非常强大的，它允许你指定一个集合，声明可以用在元素体内的集合项和索引变量。它也允许你指定开闭匹配的字符串以及在迭代中间放置分隔符。这个元素是很智能的，因此它不会偶然地附加多余的分隔符。

注意 你可以将一个 List 实例或者数组作为参数对象传给 MyBatis，当你这么做的时候，MyBatis 会自动将它包装在一个 Map 中并以名称为键。List 实例将会以“list”作为键，而数组实例的键将是“array”。

到此我们已经完成了涉及 XML 配置文件和 XML 映射文件的讨论。下一部分将详细探讨 Java API，这样才能从已创建的映射中获取最大利益。

bind

`bind` 元素可以从 OGNL 表达式中创建一个变量并将其绑定到上下文。比如：

```
<select id="selectBlogsLike" resultType="Blog">
  <bind name="pattern" value="'%' + _parameter.getTitle() + '%'" />
  SELECT * FROM BLOG
  WHERE title LIKE #{pattern}
</select>
```

Multi-db vendor support

一个配置了“_databaseId”变量的 databaseIdProvider 对于动态代码来说是可用的，这样就可以根据不同的数据库厂商构建特定的语句。比如下面的例子：

```
<insert id="insert">
  <selectKey keyProperty="id" resultType="int" order="BEFORE">
    <if test="_databaseId == 'oracle'">
      select seq_users.nextval from dual
    </if>
    <if test="_databaseId == 'db2'">
      select nextval for seq_users from sysibm.sysdummy1"
    </if>
  </selectKey>
  insert into users values (#{id}, #{name})
</insert>
```

动态 SQL 中可插拔的脚本语言

MyBatis 从 3.2 开始支持可插拔的脚本语言，因此你可以在插入一种语言的驱动（language driver）之后来写基于这种语言的动态 SQL 查询。

可以通过实现下面接口的方式来插入一种语言：

```
public interface LanguageDriver {
  ParameterHandler createParameterHandler(MappedStatement mappedSta
  SqlSource createSqlSource(Configuration configuration, XNode scri
  SqlSource createSqlSource(Configuration configuration, String scr
}
```

一旦有了自定义的语言驱动，你就可以在 mybatis-config.xml 文件中将它设置为默认语言：

```
<typeAliases>
  <typeAlias type="org.sample.MyLanguageDriver" alias="myLanguage"/
</typeAliases>
<settings>
  <setting name="defaultScriptingLanguage" value="myLanguage"/>
</settings>
```

除了设置默认语言，你也可以针对特殊的语句指定特定语言，这可以通过如下的 `lang` 属性来完成：

```
<select id="selectBlog" lang="myLanguage">
  SELECT * FROM BLOG
</select>
```

或者在你正在使用的映射中加上注解 `@Lang` 来完成：

```
public interface Mapper {
  @Lang(MyLanguageDriver.class)
  @Select("SELECT * FROM BLOG")
  List<Blog> selectBlog();
}
```

注意 可以将 Apache Velocity 作为动态语言来使用，更多细节请参考 MyBatis-Velocity 项目。

你前面看到的所有 xml 标签都是默认 MyBatis 语言提供的，它是由别名为 `xml` 语言驱动器 `org.apache.ibatis.scripting.xmltags.XmlLanguageDriver` 驱动的。

Java API

既然你已经知道如何配置 MyBatis 和创建映射文件,你就已经准备好来提升技能了。MyBatis 的 Java API 就是你收获你所做的努力的地方。正如你即将看到的,和 JDBC 相比, MyBatis 很大程度简化了你的代码而且保持简洁,很容易理解和维护。MyBatis 3 已经引入 了很多重要的改进来使得 SQL 映射更加优秀。

应用目录结构

在我们深入 Java API 之前,理解关于目录结构的最佳实践是很重要的。MyBatis 非常灵 活, 你可以用你自己的文件来做几乎所有的事情。 但是对于任一框架, 都有一些最佳的方式。

让我们看一下典型应用的目录结构:

```
/my_application
  /bin
  /devlib
  **/lib                `<-- MyBatis *.jar文件在这里。`**
  /src
    /org/myapp/
      /action
      **/data           `<-- MyBatis配置文件在这里, 包括映射器类, XML配
        /mybatis-config.xml
        /BlogMapper.java
        /BlogMapper.xml
      /model
      /service
      /view
    **/properties       `<-- 在你XML中配置的属性 文件在这里。`**
  /test
    /org/myapp/
      /action
      /data
      /model
      /service
      /view
    /properties
  /web
    /WEB-INF
      /web.xml
```

Remember, these are preferences, not requirements, but others will thank you for using a common directory structure.

这部分内容剩余的示例将假设你使用了这种目录结构。

SqlSessions

使用 MyBatis 的主要 Java 接口就是 SqlSession。尽管你可以使用这个接口执行命令,获 取映射器和管理事务。我们会讨论 SqlSession 本身更多,但是首先我们还是要了解如果获取 一个 SqlSession 实例。SqlSessions 是由 SqlSessionFactory 实例创建的。SqlSessionFactory 对 象 包 含 创 建 SqlSession 实 例 的 所 有 方 法 。 而 SqlSessionFactory 本 身 是 由 SqlSessionFactoryBuilder 创建的,它可以从 XML 配置,注解或手动配置 Java 来创建 SqlSessionFactory。

NOTE When using MyBatis with a dependency injection framework like Spring or Guice, SqlSessions are created and injected by the DI framework so you don't need to use the SqlSessionFactoryBuilder or SqlSessionFactory and can go directly to the SqlSession section. Please refer to the MyBatis-Spring or MyBatis-Guice manuals for further info.

SqlSessionFactoryBuilder

SqlSessionFactoryBuilder 有五个 build()方法,每一种都允许你从不同的资源中创建一个 SqlSession 实例。

```
SqlSessionFactory build(InputStream inputStream)
SqlSessionFactory build(InputStream inputStream, String environment
SqlSessionFactory build(InputStream inputStream, Properties propert
SqlSessionFactory build(InputStream inputStream, String env, Proper
SqlSessionFactory build(Configuration config)
```

第一种方法是最常用的,它使用了一个参照了 XML 文档或上面讨论过的更特定的 mybatis-config.xml 文件的 Reader 实例。 可选的参数是 environment 和 properties。 Environment 决定加载哪种环境,包括数据源和事务管理器。比如:

```
<environments default="development">
  <environment id="development">
    <transactionManager type="JDBC">
        ...
    <dataSource type="POOLED">
        ...
  </environment>
  <environment id="production">
    <transactionManager type="MANAGED">
        ...
    <dataSource type="JNDI">
        ...
  </environment>
</environments>
```

如果你调用了 一个使用 environment 参数 方 式的 build 方法, 那么 MyBatis 将会使用 configuration 对象来配置这个 environment。 当然, 如果你指定了一个不合法的 environment, 你会得到错误提示。 如果你调用了其中之一没有 environment 参数的 build 方法, 那么就使用 默认的 environment(在上面的示例中就会指定为 default="development")。

如果你调用了使用 properties 实例的方法,那么 MyBatis 就会加载那些 properties(属性 配置文件) ,并你在你配置中可使用它们。那些属性可以用 ${propName}语法形式多次用在 配置文件中。

回想一下,属性可以从 mybatis-config.xml 中被引用,或者直接指定它。因此理解优先级是很重要的。我们在文档前面已经提及它了,但是这里要再次重申:

如果一个属性存在于这些位置,那么 MyBatis 将会按找下面的顺序来加载它们:

- 在 properties 元素体中指定的属性首先被读取,
- 从 properties 元素的类路径 resource 或 url 指定的属性第二个被读取, 可以覆盖已经 指定的重复属性,
- 作为方法参 数传递 的属性最 后被读 取,可以 覆盖已 经从 properties 元 素体和 resource/url 属性中加载的任意重复属性。

因此,最高优先级的属性是通过方法参数传递的,之后是 resource/url 属性指定的,最后是在 properties 元素体中指定的属性。

总结一下,前四个方法很大程度上是相同的,但是由于可以覆盖,就允许你可选地指定 environment 和/或 properties。 这里给出一个从 mybatis-config.xml 文件创建 SqlSessionFactory 的示例:

```
String **resource** = "org/mybatis/builder/mybatis-config.xml";
InputStream **inputStream** = Resources.getResourceAsStream(resourc
SqlSessionFactoryBuilder **builder** = new SqlSessionFactoryBuilder
SqlSessionFactory **factory** = builder.build(inputStream);
```

注意这里我们使用了 Resources 工具类,这个类在 org.mybatis.io 包中。Resources 类正 如其名,会帮助你从类路径下,文件系统或一个 web URL 加载资源文件。看一下这个类的 源代码或者通过你的 IDE 来查看,就会看到一整套有用的方法。这里给出一个简表:

```
URL getResourceURL(String resource)
URL getResourceURL(ClassLoader loader, String resource)
InputStream getResourceAsStream(String resource)
InputStream getResourceAsStream(ClassLoader loader, String resource
Properties getResourceAsProperties(String resource)
Properties getResourceAsProperties(ClassLoader loader, String resou
Reader getResourceAsReader(String resource)
Reader getResourceAsReader(ClassLoader loader, String resource)
File getResourceAsFile(String resource)
File getResourceAsFile(ClassLoader loader, String resource)
InputStream getUrlAsStream(String urlString)
Reader getUrlAsReader(String urlString)
Properties getUrlAsProperties(String urlString)
Class classForName(String className)
```

最后一个 build 方法使用了一个 Configuration 实例。configuration 类包含你可能需要了 解 SqlSessionFactory 实例的所有内容。Configuration 类对于配置的自查很有用,包含查找和 操作 SQL 映射(不推荐使用,因为应用正接收请求) 。configuration 类有所有配置的开关, 这些你已经了解了,只在 Java API 中露出来。这里有一个简单的示例,如何手动配置 configuration 实例,然后将它传递给 build()方法来创建 SqlSessionFactory。

```
DataSource dataSource = BaseDataTest.createBlogDataSource();
TransactionFactory transactionFactory = new JdbcTransactionFactory(

Environment environment = new Environment("development", transactio

Configuration configuration = new Configuration(environment);
configuration.setLazyLoadingEnabled(true);
configuration.setEnhancementEnabled(true);
configuration.getTypeAliasRegistry().registerAlias(Blog.class);
configuration.getTypeAliasRegistry().registerAlias(Post.class);
configuration.getTypeAliasRegistry().registerAlias(Author.class);
configuration.addMapper(BoundBlogMapper.class);
configuration.addMapper(BoundAuthorMapper.class);

SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
SqlSessionFactory factory = builder.build(configuration);
```

现在你有一个 SqlSessionFactory,可以用来创建 SqlSession 实例。

SqlSessionFactory

SqlSessionFactory 有六个方法可以用来创建 SqlSession 实例。通常来说,如何决定是你 选择下面这些方法时:

- **Transaction (事务)**: 你想为 session 使用事务或者使用自动提交(通常意味着很多 数据库和/或 JDBC 驱动没有事务)?
- **Connection (连接)**: 你想 MyBatis 获得来自配置的数据源的连接还是提供你自己
- **Execution (执行)**: 你想 MyBatis 复用预处理语句和/或批量更新语句(包括插入和 删除)?

重载的 openSession()方法签名设置允许你选择这些可选中的任何一个组合。

```
SqlSession openSession()
SqlSession openSession(boolean autoCommit)
SqlSession openSession(Connection connection)
SqlSession openSession(TransactionIsolationLevel level)
SqlSession openSession(ExecutorType execType,TransactionIsolationLe
SqlSession openSession(ExecutorType execType)
SqlSession openSession(ExecutorType execType, boolean autoCommit)
SqlSession openSession(ExecutorType execType, Connection connectio
Configuration getConfiguration();
```

默认的 openSession()方法没有参数,它会创建有如下特性的 SqlSession:

- 会开启一个事务(也就是不自动提交)
- 连接对象会从由活动环境配置的数据源实例中得到。
- 事务隔离级别将会使用驱动或数据源的默认设置。
- 预处理语句不会被复用,也不会批量处理更新。

这些方法大都可以自我解释的。 开启自动提交, "true" 传递 给可选的 autoCommit 参数。 提供自定义的连接,传递一个 Connection 实例给 connection 参数。注意没有覆盖同时设置 Connection 和 autoCommit 两者的方法,因为 MyBatis 会使用当前 connection 对象提供的设 置。 MyBatis 为事务隔离级别调用使用一个 Java 枚举包装器, 称为 TransactionIsolationLevel, 否则它们按预期的方式来工作,并有 JDBC 支持的 5 级 (NONE,READ_UNCOMMITTED,READ_COMMITTED,REPEA TABLE_READ,SERIALIZA BLE)

还有一个可能对你来说是新见到的参数,就是 ExecutorType。这个枚举类型定义了 3 个 值:

- `ExecutorType.SIMPLE` : 这个执行器类型不做特殊的事情。它为每个语句的执行创建一个新的预处理语句。
- `ExecutorType.REUSE` : 这个执行器类型会复用预处理语句。
- `ExecutorType.BATCH` : 这个执行器会批量执行所有更新语句,如果 SELECT 在它们中间执行还会标定它们是 必须的,来保证一个简单并易于理解的行为。

注意 在 SqlSessionFactory 中还有一个方法我们没有提及,就是 getConfiguration()。这 个方法会返回一个 Configuration 实例,在运行时你可以使用它来自检 MyBatis 的配置。

注意 如果你已经使用之前版本 MyBatis,你要回忆那些 session,transaction 和 batch 都是分离的。现在和以往不同了,这些都包含在 session 的范围内了。你需要处理分开处理 事务或批量操作来得到它们的效果。

SqlSession

如上面所提到的,SqlSession 实例在 MyBatis 中是非常强大的一个类。在这里你会发现 所有执行语句的方法,提交或回滚事务,还有获取映射器实例。

在 SqlSession 类中有超过 20 个方法,所以将它们分开成易于理解的组合。

语句执行方法

这些方法被用来执行定义在 SQL 映射的 XML 文件中的 SELECT,INSERT,UPDA E T 和 DELETE 语句。它们都会自行解释,每一句都使用语句的 ID 属性和参数对象,参数可以 是原生类型(自动装箱或包装类) ,JavaBean,POJO 或 Map。

```
<T> T selectOne(String statement, Object parameter)
<E> List<E> selectList(String statement, Object parameter)
<K,V> Map<K,V> selectMap(String statement, Object parameter, String
int insert(String statement, Object parameter)
int update(String statement, Object parameter)
int delete(String statement, Object parameter)
```

selectOne 和 selectList 的不同仅仅是 selectOne 必须返回一个对象。 如果多余一个, 或者 没有返回 (或返回了 null) 那么就会抛出异常。 , 如果你不知道需要多少对象, 使用 selectList。

如果你想检查一个对象是否存在,那么最好返回统计数(0 或 1) 。因为并不是所有语句都需 要参数,这些方法都是有不同重载版本的,它们可以不需要参数对象。

```
<T> T selectOne(String statement)
<E> List<E> selectList(String statement)
<K,V> Map<K,V> selectMap(String statement, String mapKey)
int insert(String statement)
int update(String statement)
int delete(String statement)
```

最后,还有查询方法的三个高级版本,它们允许你限制返回行数的范围,或者提供自定义结果控制逻辑,这通常用于大量的数据集合。

```
<E> List<E> selectList (String statement, Object parameter, RowBoun
<K,V> Map<K,V> selectMap(String statement, Object parameter, String
void select (String statement, Object parameter, ResultHandler<T> h
void select (String statement, Object parameter, RowBounds rowBound
```

RowBounds 参数会告诉 MyBatis 略过指定数量的记录,还有限制返回结果的数量。RowBounds 类有一个构造方法来接收 offset 和 limit,否则是不可改变的。

```
int offset = 100;
int limit = 25;
RowBounds rowBounds = new RowBounds(offset, limit);
```

不同的驱动会实现这方面的不同级别的效率。对于最佳的表现,使用结果集类型的 SCROLL_SENSITIVE 或 SCROLL_INSENSITIVE(或句话说:不是 FORWARD_ONLY)。

ResultHandler 参数允许你按你喜欢的方式处理每一行。你可以将它添加到 List 中,创 建 Map, 或抛出每个结果而不是只保留总计。 Set 你可以使用 ResultHandler 做很多漂亮的事, 那就是 MyBatis 内部创建结果集列表。

它的接口很简单。

```
package org.apache.ibatis.session;
public interface ResultHandler<T> {
  void handleResult(ResultContext<? extends T> context);
}
```

ResultContext 参数给你访问结果对象本身的方法, 大量结果对象被创建, 你可以使用布 尔返回值的 stop()方法来停止 MyBatis 加载更多的结果。

Batch update statement Flush Method

There is method for flushing(executing) batch update statements that stored in a JDBC driver class at any timing. This method can be used when you use the `ExecutorType.BATCH` as `ExecutorType`.

```
List<BatchResult> flushStatements()
```

事务控制方法

控制事务范围有四个方法。 当然, 如果你已经选择了自动提交或你正在使用外部事务管 理器,这就没有任何效果了。然而,如果你正在使用 JDBC 事务管理员,由 Connection 实 例来控制,那么这四个方法就会派上用场:

```
void commit()
void commit(boolean force)
void rollback()
void rollback(boolean force)
```

默认情况下 MyBatis 不会自动提交事务, 除非它侦测到有插入, 更新或删除操作改变了 数据库。如果你已经做出了一些改变而没有使用这些方法,那么你可以传递 true 到 commit 和 rollback 方法来保证它会被提交(注意,你不能在自动提交模式下强制 session,或者使用 了外部事务管理器时) 。很多时候你不用调用 rollback(),因为如果你没有调用 commit 时 MyBatis 会替你完成。然而,如果你需要更多对多提交和回滚都可能的 session 的细粒度控 制,你可以使用回滚选择来使它成为可能。

NOTE MyBatis-Spring and MyBatis-Guice provide declarative transaction handling. So if you are using MyBatis with Spring or Guice please refer to their specific manuals.

清理 **Session** 级的缓存

```
void clearCache()
```

SqlSession 实例有一个本地缓存在执行 update,commit,rollback 和 close 时被清理。要 明确地关闭它(获取打算做更多的工作) ,你可以调用 clearCache()。

确保 **SqlSession** 被关闭

```
void close()
```

你必须保证的最重要的事情是你要关闭所打开的任何 session。保证做到这点的最佳方 式是下面的工作模式:

```
SqlSession session = sqlSessionFactory.openSession();
try {
    // following 3 lines pseudocod for "doing some work"
    session.insert(...);
    session.update(...);
    session.delete(...);
    session.commit();
} finally {
    session.close();
}
```

还有，如果你正在使用jdk 1.7以上的版本还有MyBatis 3.2以上的版本，你可以使用try-with-resources语句：

```
try (SqlSession session = sqlSessionFactory.openSession()) {
    // following 3 lines pseudocode for "doing some work"
    session.insert(...);
    session.update(...);
    session.delete(...);
    session.commit();
}
```

注意 就像 SqlSessionFactory,你可以通过调用 getConfiguration()方法获得 SqlSession 使用的 Configuration 实例

```
Configuration getConfiguration()
```

使用映射器

```
<T> T getMapper(Class<T> type)
```

上述的各个 insert,update,delete 和 select 方法都很强大,但也有些繁琐,没有类型安全,对于你的 IDE 也没有帮助,还有可能的单元测试。在上面的入门章节中我们已经看到 了一个使用映射器的示例。

因此, 一个更通用的方式来执行映射语句是使用映射器类。 一个映射器类就是一个简单 的接口,其中的方法定义匹配于 SqlSession 方法。下面的示例展示了一些方法签名和它们是 如何映射到 SqlSession 的。

```
public interface AuthorMapper {
  // (Author) selectOne("selectAuthor",5);
  Author selectAuthor(int id);
  // (List<Author>) selectList(“selectAuthors”)
  List<Author> selectAuthors();
  // (Map<Integer,Author>) selectMap("selectAuthors", "id")
  @MapKey("id")
  Map<Integer, Author> selectAuthors();
  // insert("insertAuthor", author)
  int insertAuthor(Author author);
  // updateAuthor("updateAuthor", author)
  int updateAuthor(Author author);
  // delete("deleteAuthor",5)
  int deleteAuthor(int id);
}
```

总之, 每个映射器方法签名应该匹配相关联的 SqlSession 方法, 而没有字符串参数 ID。 相反,方法名必须匹配映射语句的 ID。

此外,返回类型必须匹配期望的结果类型。所有常用的类型都是支持的,包括:原生类型,Map,POJO 和 JavaBean。

映射器接口不需要去实现任何接口或扩展任何类。 只要方法前面可以被用来唯一标识对 应的映射语句就可以了。

映射器接口可以扩展其他接口。当使用 XML 来构建映射器接口时要保证在合适的命名 空间中有语句。 而且, 唯一的限制就是你不能在两个继承关系的接口中有相同的方法签名 (这 也是不好的想法)。

你可以传递多个参数给一个映射器方法。 如果你这样做了, 默认情况下它们将会以它们 在参数列表中的位置来命名,比如:#{param1},#{param2}等。如果你想改变参数的名称(只在多参数 情况下) ,那么你可以在参数上使用@Param(“paramName”)注解。

你也可以给方法传递一个 RowBounds 实例来限制查询结果。

映射器注解

因为最初设计时,MyBatis 是一个 XML 驱动的框架。配置信息是基于 XML 的,而且映射语句也是定义在 XML 中的。而到了 MyBatis 3,有新的可用的选择了。MyBatis 3 构建 在基于全面而且强大的 Java 配置 API 之上。这个配置 API 是基于 XML 的 MyBatis 配置的 基础,也是新的基于注解配置的基础。注解提供了一种简单的方式来实现简单映射语句,而 不会引入大量的开销。

注意 不幸的是,Java 注解限制了它们的表现和灵活。尽管很多时间都花调查,设计和实验上,最强大的 MyBatis 映射不能用注解来构建,那并不可笑。C#属性(做示例)就没 有这些限制,因此 MyBatis.NET 将会比 XML 有更丰富的选择。也就是说,基于 Java 注解 的配置离不开它的特性。

注解有下面这些**:**

注解	目标	相对应的 **XML**	
@CacheNamespace	类	<cache>	为给 存。 flushI
@CacheNamespaceRef	类	<cacheRef>	参照 用。 的字 。
@ConstructorArgs	Method	<constructor>	收集 构造 的数
@Arg	方法	<arg> <idArg>	单 独 Cons 性: id,col id 属 属 性
			一组

@TypeDiscriminator	方法	<discriminator>	现。 jdbcT cases
@Case	方法	<case>	单独 性: va 是结 Resu 注解
@Results	方法	<resultMap>	结果 果 列 情。 Resu attrib mapp
@Result	方法	<result> <id>	在列 射。 javaT one,r 示了 中的< 单 独 似， 的， 命名
@One	方法	<association>	复 杂 selec 法)的 型的 API 注解 fet the gl laz mapp
@Many	方法	<collection>	A ma of a c sel qualif stater can lo the a fet the gl laz

			mapp join n the A the lir that c refere
`@MapKey`	方法		复 杂 性 : s 器方 适 类 在 Ja 为 Ja 用。
`@Options`	方法	映射语句的属性	这个注 宽广a 为 属 解变复 晰的方 性:us flush(result stater fetch: useG keyP 是很 定“nu Optio 支配。 免不易
`@Insert` `@Update` `@Delete` `@Select`	方法	<insert> <update> <delete> <select>	这些注 实 SC 数组 是字 的单独 Java 空间” 欢迎使 性:va 单独的
			这些可 个 类 运行 执行的 这个类

`@InsertProvider`
`@UpdateProvider`
`@DeleteProvider`
`@SelectProvider`
	方法	`<insert>`
		`<update>`
		`<delete>`
		`<select>`
			方法.
			数对
			只指
			type,
			限 定
			法名。
			Selec
			你以
			态 SC

`@Param`	`Parameter`	N/A	如果 数, 这 方法 否则, 来被 参数) 等，这 @Pa 为 #{

`@SelectKey`	`Method`	`<selectKey>`	This
			`&lt`
			for m
			`@Ins`
			`@Up`
			It is i
			you s
			anno
			any g
			via th
			confi
			state
			is the
			`key`
			prope
			that v
			value
			`true`
			SQL
			befor
			`resu`
			type
			`sta`

			This the id elem `@Se`

`@ResultMap`	`Method`	N/A	anno selec defin overr `@Co` both selec
`@ResultType`	`Method`	N/A	This a res returr have objec there @Re result the then neces anno @Se a res must @Re anno meth
`@Flush`	`Method`	N/A	If this callec `Sql` via m interf

映射申明样例

这个例子展示了如何使用 @SelectKey 注解来在插入前读取数据库序列的值：

```
@Insert("insert into table3 (id, name) values(#{nameId}, #{name})")
@SelectKey(statement="call next value for TestSequence", keyPropert
**int** insertTable3(Name name);
```

这个例子展示了如何使用 @SelectKey 注解来在插入后读取数据库识别列的值：

```
@Insert("insert into table2 (name) values(#{name})")
@SelectKey(statement="call identity()", keyProperty="nameId", befor
**int** insertTable2(Name name);
```

This example shows using the `@Flush` annotation to call the `SqlSession#flushStatements()` :

```
@Flush
List<BatchResult> flush();
```

These examples show how to name a ResultMap by specifying id attribute of @Results annotation.

```
@Results(id = "userResult", value = {
  @Result(property = "id", column = "uid", id = **true**),
  @Result(property = "firstName", column = "first_name"),
  @Result(property = "lastName", column = "last_name")
})
@Select("select * from users where id = #{id}")
User getUserById(Integer id);

@Results(id = "companyResults")
@ConstructorArgs({
  @Arg(property = "id", column = "cid", id = **true**),
  @Arg(property = "name", column = "name")
})
@Select("select * from company where id = #{id}")
Company getCompanyById(Integer id);
```

SQL语句构建器类

问题

Java程序员面对的最痛苦的事情之一就是在Java代码中嵌入SQL语句。这么来做通常是由于SQL语句需要动态来生成-否则可以将它们放到外部文件或者存储过程中。正如你已经看到的那样，MyBatis在它的XML映射特性中有一个强大的动态SQL生成方案。但有时在Java代码内部创建SQL语句也是必要的。此时，MyBatis有另外一个特性可以帮到你，在减少典型的加号,引号,新行,格式化问题和嵌入条件来处理多余的逗号或 AND 连接词之前。事实上，在Java代码中来动态生成SQL代码就是一场噩梦。例如：

```
String sql = "SELECT P.ID, P.USERNAME, P.PASSWORD, P.FULL_NAME, "
"P.LAST_NAME,P.CREATED_ON, P.UPDATED_ON " +
"FROM PERSON P, ACCOUNT A " +
"INNER JOIN DEPARTMENT D on D.ID = P.DEPARTMENT_ID " +
"INNER JOIN COMPANY C on D.COMPANY_ID = C.ID " +
"WHERE (P.ID = A.ID AND P.FIRST_NAME like ?) " +
"OR (P.LAST_NAME like ?) " +
"GROUP BY P.ID " +
"HAVING (P.LAST_NAME like ?) " +
"OR (P.FIRST_NAME like ?) " +
"ORDER BY P.ID, P.FULL_NAME";
```

The Solution

MyBatis 3提供了方便的工具类来帮助解决该问题。使用SQL类，简单地创建一个实例来调用方法生成SQL语句。上面示例中的问题就像重写SQL类那样：

```
private String selectPersonSql() {
  return new SQL() {{
    SELECT("P.ID, P.USERNAME, P.PASSWORD, P.FULL_NAME");
    SELECT("P.LAST_NAME, P.CREATED_ON, P.UPDATED_ON");
    FROM("PERSON P");
    FROM("ACCOUNT A");
    INNER_JOIN("DEPARTMENT D on D.ID = P.DEPARTMENT_ID");
    INNER_JOIN("COMPANY C on D.COMPANY_ID = C.ID");
    WHERE("P.ID = A.ID");
    WHERE("P.FIRST_NAME like ?");
    OR();
    WHERE("P.LAST_NAME like ?");
    GROUP_BY("P.ID");
    HAVING("P.LAST_NAME like ?");
    OR();
    HAVING("P.FIRST_NAME like ?");
    ORDER_BY("P.ID");
    ORDER_BY("P.FULL_NAME");
  }}.toString();
}
```

该例中有什么特殊之处？当你仔细看时，那不用担心偶然间重复出现的"AND"关键字，或者在"WHERE"和"AND"之间的选择，抑或什么都不选。该SQL类非常注意"WHERE"应该出现在何处，哪里又应该使用"AND"，还有所有的字符串链接。

SQL类

这里给出一些示例：

```
// Anonymous inner class
public String deletePersonSql() {
  return new SQL() {{
    DELETE_FROM("PERSON");
    WHERE("ID = ${id}");
  }}.toString();
}

// Builder / Fluent style
public String insertPersonSql() {
  String sql = new SQL()
    .INSERT_INTO("PERSON")
    .VALUES("ID, FIRST_NAME", "${id}, ${firstName}")
    .VALUES("LAST_NAME", "${lastName}")
    .toString();
  return sql;
}

// With conditionals (note the final parameters, required for the a
public String selectPersonLike(final String id, final String firstN
```

```
    return new SQL() {{
      SELECT("P.ID, P.USERNAME, P.PASSWORD, P.FIRST_NAME, P.LAST_NAME
      FROM("PERSON P");
      if (id != null) {
        WHERE("P.ID like ${id}");
      }
      if (firstName != null) {
        WHERE("P.FIRST_NAME like ${firstName}");
      }
      if (lastName != null) {
        WHERE("P.LAST_NAME like ${lastName}");
      }
      ORDER_BY("P.LAST_NAME");
    }}.toString();
  }

  public String deletePersonSql() {
    return new SQL() {{
      DELETE_FROM("PERSON");
      WHERE("ID = ${id}");
    }}.toString();
  }

  public String insertPersonSql() {
    return new SQL() {{
      INSERT_INTO("PERSON");
      VALUES("ID, FIRST_NAME", "${id}, ${firstName}");
      VALUES("LAST_NAME", "${lastName}");
    }}.toString();
  }

  public String updatePersonSql() {
    return new SQL() {{
      UPDATE("PERSON");
      SET("FIRST_NAME = ${firstName}");
      WHERE("ID = ${id}");
    }}.toString();
  }
```

方法	描述
`SELECT(String)`	开始或插入到 `SELECT` 子句。 可以被多次调用，参数也会添加到 `SELECT` 子句。 参数通常使用逗号分隔的列名和别名列表，但也可以是数据库驱动程序接受的任意类型。
`SELECT_DISTINCT(String)`	开始或插入到 `SELECT` 子句， 也可以插入 `DISTINCT` 关键字到生成的查询语句中。 可以被多次调用，参数也会添加到 `SELECT` 子句。 参数通常使用逗号分隔的

	列名和别名列表，但也可以是数据库驱动程序接受的任意类型。
`FROM(String)`	开始或插入到 `FROM` 子句。 可以被多次调用，参数也会添加到 `FROM` 子句。 参数通常是表名或别名，也可以是数据库驱动程序接受的任意类型。
`JOIN(String)` `INNER_JOIN(String)` `LEFT_OUTER_JOIN(String)` `RIGHT_OUTER_JOIN(String)`	基于调用的方法，添加新的合适类型的 `JOIN` 子句。 参数可以包含由列命和join on条件组合成标准的join。
`WHERE(String)`	插入新的 `WHERE` 子句条件， 由 `AND` 链接。可以多次被调用，每次都由 `AND` 来链接新条件。使用 `OR()` 来分隔 `OR` 。
`OR()`	使用 `OR` 来分隔当前的 `WHERE` 子句条件。 可以被多次调用，但在一行中多次调用或生成不稳定的 `SQL` 。
`AND()`	使用 `AND` 来分隔当前的 `WHERE` 子句条件。 可以被多次调用，但在一行中多次调用或生成不稳定的 `SQL` 。因为 `WHERE` 和 `HAVING` 二者都会自动链接 `AND` , 这是非常罕见的方法，只是为了完整性才被使用。
`GROUP_BY(String)`	插入新的 `GROUP BY` 子句元素，由逗号连接。 可以被多次调用，每次都由逗号连接新的条件。
`HAVING(String)`	插入新的 `HAVING` 子句条件。 由AND连接。可以被多次调用，每次都由 `AND` 来连接新的条件。使用 `OR()` 来分隔 `OR` .
`ORDER_BY(String)`	插入新的 `ORDER BY` 子句元素， 由逗号连接。可以多次被调用， 每次由逗号连接新的条件。
`DELETE_FROM(String)`	开始一个delete语句并指定需要从哪个表删除的表名。通常它后面都会跟着WHERE语句！
`INSERT_INTO(String)`	开始一个insert语句并指定需要插入数据的表名。后面都会跟着一个或者多个VALUES()。
`SET(String)`	针对update语句，插入到"set"列表中
`UPDATE(String)`	开始一个update语句并指定需要更新的表明。后面都会跟着一个或者多个SET()，通常也会有一个WHERE()。

`VALUES(String, String)`	插入到insert语句中。第一个参数是要插入的列名，第二个参数则是该列的值。

SqlBuilder 和 SelectBuilder (已经废弃)

在3.2版本之前，我们使用了一点不同的做法，通过实现ThreadLocal变量来掩盖一些导致Java DSL麻烦的语言限制。但这种方式已经废弃了，现代的框架都欢迎人们使用构建器类型和匿名内部类的想法。因此，SelectBuilder 和 SqlBuilder 类都被废弃了。

下面的方法仅仅适用于废弃的SqlBuilder 和 SelectBuilder 类。

方法	描述
`BEGIN()` / `RESET()`	这些方法清空SelectBuilder类的ThreadLocal状态，并且准备一个新的构建语句。开始新的语句时， `BEGIN()` 读取得最好。 由于一些原因（在某些条件下，也许是逻辑需要一个完全不同的语句），在执行中清理语句 `RESET()` 读取得最好。
`SQL()`	返回生成的 `SQL()` 并重置 `SelectBuilder` 状态 (好像 `BEGIN()` 或 `RESET()` 被调用了). 因此，该方法只能被调用一次！

SelectBuilder 和 SqlBuilder 类并不神奇，但是知道它们如何工作也是很重要的。SelectBuilder 使用 SqlBuilder 使用了静态导入和ThreadLocal变量的组合来开启整洁语法，可以很容易地和条件交错。使用它们，静态导入类的方法即可，就像这样(一个或其它，并非两者):

```
import static org.apache.ibatis.jdbc.SelectBuilder.*;
```

```
import static org.apache.ibatis.jdbc.SqlBuilder.*;
```

这就允许像下面这样来创建方法：

```
/* DEPRECATED */
public String selectBlogsSql() {
  BEGIN(); // Clears ThreadLocal variable
  SELECT("*");
  FROM("BLOG");
  return SQL();
}
```

```
/* DEPRECATED */
private String selectPersonSql() {
  BEGIN(); // Clears ThreadLocal variable
  SELECT("P.ID, P.USERNAME, P.PASSWORD, P.FULL_NAME");
  SELECT("P.LAST_NAME, P.CREATED_ON, P.UPDATED_ON");
  FROM("PERSON P");
  FROM("ACCOUNT A");
  INNER_JOIN("DEPARTMENT D on D.ID = P.DEPARTMENT_ID");
  INNER_JOIN("COMPANY C on D.COMPANY_ID = C.ID");
  WHERE("P.ID = A.ID");
  WHERE("P.FIRST_NAME like ?");
  OR();
  WHERE("P.LAST_NAME like ?");
  GROUP_BY("P.ID");
  HAVING("P.LAST_NAME like ?");
  OR();
  HAVING("P.FIRST_NAME like ?");
  ORDER_BY("P.ID");
  ORDER_BY("P.FULL_NAME");
  return SQL();
}
```

Logging

Mybatis内置的日志工厂提供日志功能，具体的日志实现有以下几种工具：

- SLF4J
- Apache Commons Logging
- Log4j 2
- Log4j
- JDK logging

具体选择哪个日志实现工具由MyBatis的内置日志工厂确定。它会使用最先找到的（按上文列举的顺序查找）。 如果一个都未找到，日志功能就会被禁用。

不少应用服务器的classpath中已经包含Commons Logging，如Tomcat和WebShpere， 所以MyBatis会把它作为具体的日志实现。记住这点非常重要。这将意味着，在诸如 WebSphere的环境中——WebSphere提供了Commons Logging的私有实现，你的Log4J配置将被忽略。 这种做法不免让人悲催，MyBatis怎么能忽略你的配置呢？事实上，因Commons Logging已经存 在了，按照优先级顺序，Log4J自然就被忽略了！不过，如果你的应用部署在一个包含Commons Logging的环境， 而你又想用其他的日志框架，你可以根据需要调用如下的某一方法：

```
org.apache.ibatis.logging.LogFactory.useSlf4jLogging();
org.apache.ibatis.logging.LogFactory.useLog4JLogging();
org.apache.ibatis.logging.LogFactory.useJdkLogging();
org.apache.ibatis.logging.LogFactory.useCommonsLogging();
org.apache.ibatis.logging.LogFactory.useStdOutLogging();
```

如果的确需要调用以上的某个方法，请在其他所有MyBatis方法之前调用它。另外，只有在相应日志实现中存在 的前提下，调用对应的方法才是有意义的，否则MyBatis一概忽略。如你环境中并不存在Log4J，你却调用了 相应的方法，MyBatis就会忽略这一调用，代之默认的查找顺序查找日志实现。

关于SLF4J、Apache Commons Logging、Apache Log4J和JDK Logging的API介绍已经超出本文档的范围。 不过，下面的例子可以作为一个快速入门。关于这些日志框架的更多信息，可以参考以下链接：

- Apache Commons Logging
- Apache Log4j
- JDK Logging API

Logging Configuration

MyBatis可以对包、 类、命名空间和全限定的语句记录日志。

具体怎么做，视使用的日志框架而定，这里以Log4J为例。配置日志功能非常简单：添加几个配置文件， 如log4j.properties,再添加个jar包，如log4j.jar。下面是具体的例子，共两个步骤：

步骤1： 添加 Log4J 的 jar 包

因为采用Log4J，要确保在应用中对应的jar包是可用的。要满足这一点，只要将jar包添加到应用的classpath中即可。 Log4J的jar包可以从上面的链接中下载。

具体而言，对于web或企业应用，需要将 `log4j.jar` 添加到 `WEB-INF/lib` 目录； 对于独立应用， 可以将它添加到jvm的 `-classpath` 启动参数中。

步骤2：配置Log4J

配置Log4J比较简单， 比如需要记录这个mapper接口的日志:

```
package org.mybatis.example;
public interface BlogMapper {
  @Select("SELECT * FROM blog WHERE id = #{id}")
  Blog selectBlog(int id);
}
```

只要在应用的classpath中创建一个名称为 `log4j.properties` 的文件， 文件的具体内容如下：

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout
# MyBatis logging configuration...
log4j.logger.org.mybatis.example.BlogMapper=TRACE
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

添加以上配置后，Log4J就会把 `org.mybatis.example.BlogMapper` 的详细执行日志记录下来，对于应用中的其它类则仅仅记录错误信息。

也可以将日志从整个mapper接口级别调整到到语句级别，从而实现更细粒度的控制。如下配置只记录 `selectBlog` 语句的日志：

```
log4j.logger.org.mybatis.example.BlogMapper.selectBlog=TRACE
```

与此相对，可以对一组mapper接口记录日志，只要对mapper接口所在的包开启日志功能即可：

```
log4j.logger.org.mybatis.example=TRACE
```

某些查询可能会返回大量的数据，只想记录其执行的SQL语句该怎么办？为此，Mybatis中SQL语 句的日志级别被设为DEBUG（JDK Logging中为FINE），结果日志的级别为TRACE（JDK Logging中为FINER)。所以，只要将日志级别调整为DEBUG即可达到目的：

```
log4j.logger.org.mybatis.example=DEBUG
```

要记录日志的是类似下面的mapper文件而不是mapper接口又该怎么呢？

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = #{id}
  </select>
</mapper>
```

对这个文件记录日志，只要对命名空间增加日志记录功能即可：

```
log4j.logger.org.mybatis.example.BlogMapper=TRACE
```

进一步，要记录具体语句的日志可以这样做：

```
log4j.logger.org.mybatis.example.BlogMapper.selectBlog=TRACE
```

看到了吧，两种配置没差别！

配置文件 `log4j.properties` 的余下内容是针对日志格式的，这一内容已经超出本 文档范围。关于Log4J的更多内容，可以参考Log4J的网站。不过，可以简单试一下看看，不同的配置 会产生什么不一样的效果。

项目文档

项目总体信息

本文档提供了与本项目相关信息的各种文档和链接的概述。这里的所有内容都是根据本项目信息生成的，自动生成过程依靠 Maven 。

概述

文档	描述
持续集成	持续集成是以一定的频率和固定的基础进行代码构建和测试的过程，这里列出了所有的持续集成过程。
项目依赖	这一文档列出了项目的依赖并提供了每个依赖的相关信息。
Dependency Information	This document describes how to to include this project as a dependency using various dependency management tools.
Distribution Management	This document provides informations on the distribution management of this project.
问题跟踪	这是本项目中的问题管理系统的链接。用户可以在这里进行问题(故障、特性、变更请求)的创建和查询。
项目授权	这是关于本项目授权的定义。
邮件列表	本文档提供了本项目的邮件列表的订阅和归档信息。
Plugin Management	This document lists the plugins that are defined through pluginManagement.
Project Plugins	This document lists the build plugins and the report plugins used by this project.
项目团队	本文档提供了本项目中成员的信息.他们是在本项目中以某种形式做出了贡献的个人.
源代码库	这是一个在线代码库,它可以通过Web浏览器进行查看.
项目概要	此文档列出了该项目其它的相关信息

概述

本项目使用 Travis CI。

访问

以下是在本项目中使用的持续集成系统。

```
[https://travis-ci.org/mybatis/mybatis-3/](https://travis-ci.org/my
```

提醒方法

未定义任何的提醒方法，请以后再查看。

项目依赖

compile

以下是这一项目的编译依赖列表。编译和运行本应用需要这些依赖。

GroupId	ArtifactId	版本	类型	License	备注
cglib	cglib	3.1	jar	ASF 2.0	Yes
commons-logging	commons-logging	1.2	jar	The Apache Software License, Version 2.0	Yes
log4j	log4j	1.2.17	jar	The Apache Software License, Version 2.0	Yes
ognl	ognl	3.0.11	jar	The Apache Software License, Version 2.0	Yes
org.apache.logging.log4j	log4j-core	2.3	jar	The Apache Software License, Version 2.0	Yes
org.javassist	javassist	3.20.0-GA	jar	MPL 1.1-LGPL 2.1-Apache License 2.0	Yes
org.slf4j	slf4j-api	1.7.12	jar	MIT License	Yes
org.slf4j	slf4j-log4j12	1.7.12	jar	MIT License	Yes

test

以下是这一项目的测试依赖列表。这些依赖仅仅在编译和运行本应用中的单元测试时需要。

GroupId	ArtifactId	版本	类型	License
commons-dbcp	commons-dbcp	1.4	jar	The Apache Software License, Version 2.0
javax.transaction	transaction-api	1.1	jar	-
junit	junit	4.12	jar	Eclipse Public License 1.0
org.apache.derby	derby	10.11.1.1	jar	Apache 2
org.apache.velocity	velocity	1.7	jar	The Apache Software License, Version 2.0
org.hsqldb	hsqldb	2.3.2	jar	HSQLDB License, a BSD open source license
org.mockito	mockito-core	1.10.19	jar	The MIT License
postgresql	postgresql	9.1-901-1.jdbc4	jar	BSD License

Dependency Information

Apache Maven

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.4.0-SNAPSHOT</version>
</dependency>
```

Apache Buildr

```
'org.mybatis:mybatis:jar:3.4.0-SNAPSHOT'
```

Apache Ivy

```
<dependency org="org.mybatis" name="mybatis" rev="3.4.0-SNAPSHOT">
  <artifact name="mybatis" type="jar" />
</dependency>
```

Groovy Grape

```
@Grapes(
@Grab(group='org.mybatis', module='mybatis', version='3.4.0-SNAPSHO
)
```

Grails

```
compile 'org.mybatis:mybatis:3.4.0-SNAPSHOT'
```

Leiningen

```
[org.mybatis/mybatis "3.4.0-SNAPSHOT"]
```

SBT

```
libraryDependencies += "org.mybatis" % "mybatis" % "3.4.0-SNAPSHOT"
```

Overview

The following is the distribution management information used by this project.

Repository - sonatype-nexus-staging

https://oss.sonatype.org/service/local/staging/deploy/maven2/

Snapshot Repository - sonatype-nexus-snapshots

https://oss.sonatype.org/content/repositories/snapshots/

Site - gh-pages

git:ssh://git@github.com/mybatis/mybatis-3.git?gh-pages#

概述

本项目使用GitHub Issue Management管理问题。

问题跟踪

问题、故障和新特性请求请通过下述的本项目的问题跟踪系统提交。

```
[https://github.com/mybatis/mybatis-3/issues](https://github.com/my
```

概述

一般来说，项目中所列的授权仅对项目自身而言，不包括项目的依赖。

项目授权

The Apache Software License, Version 2.0

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1\. Definitions.

   "License" shall mean the terms and conditions for use, reproc
   and distribution as defined by Sections 1 through 9 of this c

   "Licensor" shall mean the copyright owner or entity authorize
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and
   other entities that control, are controlled by, or are under
   control with that entity. For the purposes of this definition
   "control" means (i) the power, direct or indirect, to cause t
   direction or management of such entity, whether by contract c
   otherwise, or (ii) ownership of fifty percent (50%) or more c
   outstanding shares, or (iii) beneficial ownership of such ent

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifi
   including but not limited to software source code, documentat
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source c
   Object form, made available under the License, as indicated b
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).
```

```
      "Derivative Works" shall mean any work, whether in Source or
      form, that is based on (or derived from) the Work and for whi
      editorial revisions, annotations, elaborations, or other modi
      represent, as a whole, an original work of authorship. For th
      of this License, Derivative Works shall not include works tha
      separable from, or merely link (or bind by name) to the inter
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or add
      to that Work or Derivative Works thereof, that is intentional
      submitted to Licensor for inclusion in the Work by the copyri
      or by an individual or Legal Entity authorized to submit on b
      the copyright owner. For the purposes of this definition, "su
      means any form of electronic, verbal, or written communicatio
      to the Licensor or its representatives, including but not lim
      communication on electronic mailing lists, source code contro
      and issue tracking systems that are managed by, or on behalf
      Licensor for the purpose of discussing and improving the Work
      excluding communication that is conspicuously marked or other
      designated in writing by the copyright owner as "Not a Contri

      "Contributor" shall mean Licensor and any individual or Legal
      on behalf of whom a Contribution has been received by Licenso
      subsequently incorporated within the Work.

   2\. Grant of Copyright License. Subject to the terms and conditi
      this License, each Contributor hereby grants to You a perpetu
      worldwide, non-exclusive, no-charge, royalty-free, irrevocabl
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribut
      Work and such Derivative Works in Source or Object form.

   3\. Grant of Patent License. Subject to the terms and conditions
      this License, each Contributor hereby grants to You a perpetu
      worldwide, non-exclusive, no-charge, royalty-free, irrevocabl
      (except as stated in this section) patent license to make, ha
      use, offer to sell, sell, import, and otherwise transfer the
      where such license applies only to those patent claims licens
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution
      with the Work to which such Contribution(s) was submitted. If
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the W
      or a Contribution incorporated within the Work constitutes di
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall termina
      as of the date such litigation is filed.

   4\. Redistribution. You may reproduce and distribute copies of t
      Work or Derivative Works thereof in any medium, with or witho
      modifications, and in Source or Object form, provided that Yo
      meet the following conditions:
```

```
    (a) You must give any other recipients of the Work or
        Derivative Works a copy of this License; and

    (b) You must cause any modified files to carry prominent noti
        stating that You changed the files; and

    (c) You must retain, in the Source form of any Derivative Wor
        that You distribute, all copyright, patent, trademark, ar
        attribution notices from the Source form of the Work,
        excluding those notices that do not pertain to any part c
        the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part of its
        distribution, then any Derivative Works that You distribu
        include a readable copy of the attribution notices contai
        within such NOTICE file, excluding those notices that do
        pertain to any part of the Derivative Works, in at least
        of the following places: within a NOTICE text file distri
        as part of the Derivative Works; within the Source form c
        documentation, if provided along with the Derivative Work
        within a display generated by the Derivative Works, if ar
        wherever such third-party notices normally appear. The cc
        of the NOTICE file are for informational purposes only ar
        do not modify the License. You may add Your own attributi
        notices within Derivative Works that You distribute, alor
        or as an addendum to the NOTICE text from the Work, provi
        that such additional attribution notices cannot be constr
        as modifying the License.

    You may add Your own copyright statement to Your modificatior
    may provide additional or different license terms and conditi
    for use, reproduction, or distribution of Your modifications,
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies
    the conditions stated in this License.

 5\. Submission of Contributions. Unless You explicitly state oth
    any Contribution intentionally submitted for inclusion in the
    by You to the Licensor shall be under the terms and conditior
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or
    the terms of any separate license agreement you may have exec
    with Licensor regarding such Contributions.

 6\. Trademarks. This License does not grant permission to use th
    names, trademarks, service marks, or product names of the Lic
    except as required for reasonable and customary use in descri
    origin of the Work and reproducing the content of the NOTICE

 7\. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
```

```
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
      implied, including, without limitation, any warranties or co
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determinin
      appropriateness of using or redistributing the Work and assum
      risks associated with Your exercise of permissions under this

   8\. Limitation of Liability. In no event and under no legal theo
      whether in tort (including negligence), contract, or otherwis
      unless required by applicable law (such as deliberate and gro
      negligent acts) or agreed to in writing, shall any Contributo
      liable to You for damages, including any direct, indirect, sp
      incidental, or consequential damages of any character arising
      result of this License or out of the use or inability to use
      Work (including but not limited to damages for loss of goodwi
      work stoppage, computer failure or malfunction, or any and al
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9\. Accepting Warranty or Additional Liability. While redistribu
      the Work or Derivative Works thereof, You may choose to offer
      and charge a fee for, acceptance of support, warranty, indemn
      or other liability obligations and/or rights consistent with
      License. However, in accepting such obligations, You may act
      on Your own behalf and on Your sole responsibility, not on be
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the followin
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't inclu
      the brackets!)  The text should be enclosed in the appropriat
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright [yyyy] [name of copyright owner]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, softw
   distributed under the License is distributed on an "AS IS" BASIS
```

```
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    See the License for the specific language governing permissions
    limitations under the License.
```

The Apache Software License, Version 2.0

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

    TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

    1\. Definitions.

       "License" shall mean the terms and conditions for use, reproc
       and distribution as defined by Sections 1 through 9 of this d

       "Licensor" shall mean the copyright owner or entity authorize
       the copyright owner that is granting the License.

       "Legal Entity" shall mean the union of the acting entity and
       other entities that control, are controlled by, or are under
       control with that entity. For the purposes of this definitior
       "control" means (i) the power, direct or indirect, to cause t
       direction or management of such entity, whether by contract c
       otherwise, or (ii) ownership of fifty percent (50%) or more c
       outstanding shares, or (iii) beneficial ownership of such ent

       "You" (or "Your") shall mean an individual or Legal Entity
       exercising permissions granted by this License.

       "Source" form shall mean the preferred form for making modifi
       including but not limited to software source code, documentat
       source, and configuration files.

       "Object" form shall mean any form resulting from mechanical
       transformation or translation of a Source form, including but
       not limited to compiled object code, generated documentation,
       and conversions to other media types.

       "Work" shall mean the work of authorship, whether in Source c
       Object form, made available under the License, as indicated b
       copyright notice that is included in or attached to the work
       (an example is provided in the Appendix below).

       "Derivative Works" shall mean any work, whether in Source or
       form, that is based on (or derived from) the Work and for whi
       editorial revisions, annotations, elaborations, or other modi
       represent, as a whole, an original work of authorship. For th
       of this License, Derivative Works shall not include works tha
```

```
      separable from, or merely link (or bind by name) to the inter
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or add
      to that Work or Derivative Works thereof, that is intentional
      submitted to Licensor for inclusion in the Work by the copyri
      or by an individual or Legal Entity authorized to submit on b
      the copyright owner. For the purposes of this definition, "su
      means any form of electronic, verbal, or written communicatio
      to the Licensor or its representatives, including but not lim
      communication on electronic mailing lists, source code contro
      and issue tracking systems that are managed by, or on behalf
      Licensor for the purpose of discussing and improving the Work
      excluding communication that is conspicuously marked or other
      designated in writing by the copyright owner as "Not a Contri

      "Contributor" shall mean Licensor and any individual or Legal
      on behalf of whom a Contribution has been received by Licenso
      subsequently incorporated within the Work.

   2\. Grant of Copyright License. Subject to the terms and conditi
      this License, each Contributor hereby grants to You a perpetu
      worldwide, non-exclusive, no-charge, royalty-free, irrevocabl
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribut
      Work and such Derivative Works in Source or Object form.

   3\. Grant of Patent License. Subject to the terms and conditions
      this License, each Contributor hereby grants to You a perpetu
      worldwide, non-exclusive, no-charge, royalty-free, irrevocabl
      (except as stated in this section) patent license to make, ha
      use, offer to sell, sell, import, and otherwise transfer the
      where such license applies only to those patent claims licens
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contributio
      with the Work to which such Contribution(s) was submitted. If
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the W
      or a Contribution incorporated within the Work constitutes di
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall termina
      as of the date such litigation is filed.

   4\. Redistribution. You may reproduce and distribute copies of t
      Work or Derivative Works thereof in any medium, with or witho
      modifications, and in Source or Object form, provided that Yo
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent noti
```

```
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Wor
          that You distribute, all copyright, patent, trademark, an
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part o
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribu
          include a readable copy of the attribution notices contai
          within such NOTICE file, excluding those notices that do
          pertain to any part of the Derivative Works, in at least
          of the following places: within a NOTICE text file distri
          as part of the Derivative Works; within the Source form o
          documentation, if provided along with the Derivative Work
          within a display generated by the Derivative Works, if an
          wherever such third-party notices normally appear. The co
          of the NOTICE file are for informational purposes only an
          do not modify the License. You may add Your own attributi
          notices within Derivative Works that You distribute, alon
          or as an addendum to the NOTICE text from the Work, provi
          that such additional attribution notices cannot be constr
          as modifying the License.

      You may add Your own copyright statement to Your modification
      may provide additional or different license terms and conditi
      for use, reproduction, or distribution of Your modifications,
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies
      the conditions stated in this License.

   5\. Submission of Contributions. Unless You explicitly state oth
      any Contribution intentionally submitted for inclusion in the
      by You to the Licensor shall be under the terms and condition
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or
      the terms of any separate license agreement you may have exec
      with Licensor regarding such Contributions.

   6\. Trademarks. This License does not grant permission to use th
      names, trademarks, service marks, or product names of the Lic
      except as required for reasonable and customary use in descri
      origin of the Work and reproducing the content of the NOTICE

   7\. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
      implied, including, without limitation, any warranties or con
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determinin
      appropriateness of using or redistributing the Work and assum
```

```
      risks associated with Your exercise of permissions under this

   8\. Limitation of Liability. In no event and under no legal theo
      whether in tort (including negligence), contract, or otherwis
      unless required by applicable law (such as deliberate and gro
      negligent acts) or agreed to in writing, shall any Contributo
      liable to You for damages, including any direct, indirect, sp
      incidental, or consequential damages of any character arising
      result of this License or out of the use or inability to use
      Work (including but not limited to damages for loss of goodwi
      work stoppage, computer failure or malfunction, or any and al
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9\. Accepting Warranty or Additional Liability. While redistribu
      the Work or Derivative Works thereof, You may choose to offer
      and charge a fee for, acceptance of support, warranty, indemn
      or other liability obligations and/or rights consistent with
      License. However, in accepting such obligations, You may act
      on Your own behalf and on Your sole responsibility, not on be
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the followin
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't includ
      the brackets!)  The text should be enclosed in the appropriat
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright [yyyy] [name of copyright owner]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, softw
   distributed under the License is distributed on an "AS IS" BASIS
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   See the License for the specific language governing permissions
   limitations under the License.
```

项目邮件列表

这些是本项目建立的邮件列表。每个列表都有一个订阅、退订和归档的链接。

名称	订阅	取消订阅	发送邮件	归档	其他归档
mybatis-dev	订阅	取消订阅	发送邮件	groups.google.com	-
mybatis-user	订阅	取消订阅	发送邮件	groups.google.com	mybatis-user.963551.n3.nabble.com
mybatis-commits	订阅	取消订阅	发送邮件	groups.google.com	-

Project Plugin Management

GroupId	ArtifactId	Version
com.mycila	license-maven-plugin	2.11
org.apache.maven.plugins	maven-antrun-plugin	1.8
org.apache.maven.plugins	maven-assembly-plugin	2.5.5
org.apache.maven.plugins	maven-clean-plugin	2.6.1
org.apache.maven.plugins	maven-compiler-plugin	3.3
org.apache.maven.plugins	maven-dependency-plugin	2.10
org.apache.maven.plugins	maven-deploy-plugin	2.8.2
org.apache.maven.plugins	maven-install-plugin	2.5.2
org.apache.maven.plugins	maven-pdf-plugin	1.3
org.apache.maven.plugins	maven-release-plugin	2.5.2
org.apache.maven.plugins	maven-resources-plugin	2.7
org.apache.maven.plugins	maven-shade-plugin	2.4
org.apache.maven.plugins	maven-site-plugin	3.4
org.apache.maven.plugins	maven-surefire-plugin	2.18.1
org.codehaus.mojo	versions-maven-plugin	2.2
org.jacoco	http://jacoco-maven-plugin	0.7.5.201505241946

Project Build Plugins

GroupId	ArtifactId	Version
com.mycila	license-maven-plugin	2.11
org.apache.felix	maven-bundle-plugin	2.5.4
org.apache.maven.plugins	maven-clean-plugin	2.6.1
org.apache.maven.plugins	maven-compiler-plugin	3.3
org.apache.maven.plugins	maven-deploy-plugin	2.8.2
org.apache.maven.plugins	maven-enforcer-plugin	1.4
org.apache.maven.plugins	maven-install-plugin	2.5.2
org.apache.maven.plugins	maven-jar-plugin	2.6
org.apache.maven.plugins	maven-pdf-plugin	1.3
org.apache.maven.plugins	maven-resources-plugin	2.7
org.apache.maven.plugins	maven-scm-plugin	1.9.4
org.apache.maven.plugins	maven-shade-plugin	2.4
org.apache.maven.plugins	maven-site-plugin	3.4
org.apache.maven.plugins	maven-surefire-plugin	2.18.1
org.codehaus.mojo	animal-sniffer-maven-plugin	1.14
org.codehaus.mojo	clirr-maven-plugin	2.6.1
org.jacoco	http://jacoco-maven-plugin	0.7.5.201505241946

Project Report Plugins

GroupId	ArtifactId	Version
org.apache.maven.plugins	maven-changes-plugin	2.11
org.apache.maven.plugins	maven-javadoc-plugin	2.10.3
org.apache.maven.plugins	maven-jxr-plugin	2.5
org.apache.maven.plugins	maven-pmd-plugin	3.4
org.apache.maven.plugins	maven-project-info-reports-plugin	2.8
org.apache.maven.plugins	maven-surefire-report-plugin	2.18.1
org.codehaus.mojo	clirr-maven-plugin	2.6.1
org.codehaus.mojo	findbugs-maven-plugin	3.0.1
org.codehaus.mojo	jdepend-maven-plugin	2.0
org.codehaus.mojo	taglist-maven-plugin	2.4
org.codehaus.mojo	versions-maven-plugin	2.2
org.jacoco	http://jacoco-maven-plugin	0.7.5.201505241946

团队

一个成功的项目要求许多人扮演许多的角色。其中一些成员编写代码或者文档，同时其他成员则通过测试、打补丁以及提建议等方式实现自己的价值。

一个团队由团队成员以及贡献者组成。团队成员直接访问项目的源代码并积极地参与编码工作。贡献者则通过向成员提交补丁和提出建议来完善项目。一个项目中贡献者的数量是不限的。请立刻加入到贡献行列中来吧。衷心感谢所有对项目做出贡献的人。

成员

以下是开发者的列表，他们有提交的权限，在项目中以某种方式直接做出了贡献。

Image	Id	姓名	电子邮件
	cbegin	Clinton Begin	clinton.begin@gmail.com
	brandon.goodin	Brandon Goodin	brandon.goodin@gmail.co
	mentat6114	Christian Poitras	christian.poitras@ircm.qc.c
	emacarron	Eduardo Macarron	eduardo.macarron@gmail.
	mnesarco	Frank Martinez	mnesarco@gmail.com
	agustafson	Andrew Gustafson	gus4000@gmail.com
	hpresnall	Hunter Presnall	hpresnall@gmail.com
	harawata	Iwao Ave	harawata@gmail.com
	jeffgbutler	Jeff Butler	jeffgbutler@gmail.com

	hazendaz	Jeremy Landis	jeremylandis@hotmail.com
	nospam@kaigrabfelder.de	Kai Grabfelder	nospam@kaigrabfelder.de
	lmeadors	Larry Meadors	larry.meadors@gmail.com
	marcosperanza	Marco Speranza	marco.speranza79@gmail
	nmaves	Nathan Maves	nathan.maves@gmail.com
	putthibongb	Putthibong Boonbong	putthibongb@gmail.com
	simonetripodi	Simone Tripodi	simone.tripodi@gmail.com
	h3adache	Tim Chen	chengt@gmail.com

贡献者

以下的其他人通过建议，补丁或者文档对本项目做出了贡献。

Image	姓名	电子邮件
	Adam Gent	adam.gent@evocatus.com
	Andrea Selva	selva.andre@gmail.com
	Antonio Sánchez	juntandolineas@gmail.com
	Arkadi Shishlov	arkadi.shishlov@gmail.com

	Axel Doerfler	axel.doerfler@gmail.com
	Chris Dadej	chris.dadej@gmail.com
	Denis Vygovskiy	qizant@gmail.com
	Franta Mejta	mejta@rewor.cz
	Jurriaan Pruys	jurriaan@pruys.com
	Keith Wong	wongkwl@gmail.com
	Lasse Voss	lasse.voss@motor-talk-gmbh.de
	Luke Stevens	nosuchluke@gmail.com
	Paul Krause	paulkrause88@alum.mit.edu
	Peter Leibiger	kuhnroyal@gmail.com
	Riccardo Cossu	riccardo.cossu@gmail.com
	Tomá? Neuberg	neuberg@m-atelier.cz

成员

以下是开发者的列表，他们有提交的权限，在项目中以某种方式直接做出了贡献。

	cbegin	Clinton Begin	clinton.begin@gmail.com
	brandon.goodin	Brandon Goodin	brandon.goodin@gmail.co
	mentat6114	Christian Poitras	christian.poitras@ircm.qc.
	emacarron	Eduardo Macarron	eduardo.macarron@gmail.
	mnesarco	Frank Martinez	mnesarco@gmail.com
	agustafson	Andrew Gustafson	gus4000@gmail.com
	hpresnall	Hunter Presnall	hpresnall@gmail.com
	harawata	Iwao Ave	harawata@gmail.com
	jeffgbutler	Jeff Butler	jeffgbutler@gmail.com
	hazendaz	Jeremy Landis	jeremylandis@hotmail.com
	nospam@kaigrabfelder.de	Kai Grabfelder	nospam@kaigrabfelder.de
	lmeadors	Larry Meadors	larry.meadors@gmail.com
	marcosperanza	Marco Speranza	marco.speranza79@gmail
	nmaves	Nathan Maves	nathan.maves@gmail.com
	putthibongb	Putthibong Boonbong	putthibongb@gmail.com

	simonetripodi	Simone Tripodi	simone.tripodi@gmail.com
	h3adache	Tim Chen	chengt@gmail.com

贡献者

以下的其他人通过建议，补丁或者文档对本项目做出了贡献。

Image	姓名	电子邮件
	Adam Gent	adam.gent@evocatus.com
	Andrea Selva	selva.andre@gmail.com
	Antonio Sánchez	juntandolineas@gmail.com
	Arkadi Shishlov	arkadi.shishlov@gmail.com
	Axel Doerfler	axel.doerfler@gmail.com
	Chris Dadej	chris.dadej@gmail.com
	Denis Vygovskiy	qizant@gmail.com
	Franta Mejta	mejta@rewor.cz
	Jurriaan Pruys	jurriaan@pruys.com
	Keith Wong	wongkwl@gmail.com

	Keith Wong	wongkwl@gmail.com
	Lasse Voss	lasse.voss@motor-talk-gmbh.de
	Luke Stevens	nosuchluke@gmail.com
	Paul Krause	paulkrause88@alum.mit.edu
	Peter Leibiger	kuhnroyal@gmail.com
	Riccardo Cossu	riccardo.cossu@gmail.com
	Tomá? Neuberg	neuberg@m-atelier.cz

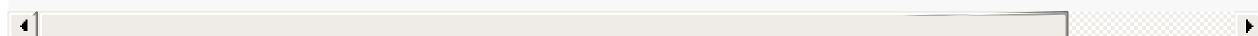
概述

This project uses Git to manage its source code. Instructions on Git use can be found at http://git-scm.com/documentation.

Web访问

以下是在线源代码库的链接。

```
[http://github.com/mybatis/mybatis-3](http://github.com/mybatis/myb
```



匿名访问

The source can be checked out anonymously from Git with this command (See http://git-scm.com/docs/git-clone):

```
$ git clone ssh://github.com/mybatis/mybatis-3.git
```

开发者访问

Only project developers can access the Git tree via this method (See http://git-scm.com/docs/git-clone).

```
$ git clone ssh://git@github.com/mybatis/mybatis-3.git
```

通过防火墙访问

请参考使用的源代码管理工具的文档以获得更多的通过防火墙进行访问的信息。

项目概要

项目信息

字段	值
名称	mybatis
描述	The MyBatis data mapper framework makes it easier to use a relational database with object-oriented applications. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations. Simplicity is the biggest advantage of the MyBatis data mapper over object relational mapping tools.
主页	http://www.mybatis.org/core/

项目组织

字段	值
名称	MyBatis.org
URL	http://www.mybatis.org/

构建信息

字段	值
GroupId	org.mybatis
ArtifactId	mybatis
版本	3.4.0-SNAPSHOT
类型	jar
Java Version	1.6

项目信息

字段	值
名称	mybatis
描述	The MyBatis data mapper framework makes it easier to use a relational database with object-oriented applications. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations. Simplicity is the biggest advantage of the MyBatis data mapper over object relational mapping tools.
主页	http://www.mybatis.org/core/

项目组织

字段	值
名称	MyBatis.org
URL	http://www.mybatis.org/

构建信息

字段	值
GroupId	org.mybatis
ArtifactId	mybatis
版本	3.4.0-SNAPSHOT
类型	jar
Java Version	1.6

生成报表

本文档提供了对项目中各种报表的概述，这些报表生成自 Maven 每一个报表在以下简要介绍。

概述

文档	描述
JavaDocs	JavaDoc API documentation.
Source Xref	HTML based, cross-reference version of Java source code.
Test Source Xref	HTML based, cross-reference version of Java test source code.
JDepend	JDepend traverses Java class file directories and generates design quality metrics for each Java package. JDepend allows you to automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to manage package dependencies effectively.
FindBugs	Generates a source code report with the FindBugs Library.
Surefire Report	Report on the test results of the project.
CPD	Duplicate code detection.
PMD	Verification of coding rules.
Tag List	Report on various tags found in the code.
Clirr	Report on binary and source API differences between releases
Dependency Updates Report	Provides details of the dependencies which have updated versions available.
Plugin Updates Report	Provides details of the plugins used by this project which have newer versions available.
Property Updates Report	Provides details of properties which control versions of dependencies and/or plugins, and indicates any newer versions which are available.

概述

文档	描述
JavaDocs	JavaDoc API documentation.
Source Xref	HTML based, cross-reference version of Java source code.
Test Source Xref	HTML based, cross-reference version of Java test source code.
JDepend	JDepend traverses Java class file directories and generates design quality metrics for each Java package. JDepend allows you to automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to manage package dependencies effectively.
FindBugs	Generates a source code report with the FindBugs Library.
Surefire Report	Report on the test results of the project.
CPD	Duplicate code detection.
PMD	Verification of coding rules.
Tag List	Report on various tags found in the code.
Clirr	Report on binary and source API differences between releases
Dependency Updates Report	Provides details of the dependencies which have updated versions available.
Plugin Updates Report	Provides details of the plugins used by this project which have newer versions available.
Property Updates Report	Provides details of properties which control versions of dependencies and/or plugins, and indicates any newer versions which are available.